On Software Security of Building Automation Systems

Christopher Morales-Gonzalez*, Matthew Harper*, Boya Yuan†, Xinwen Fu*
*University of Massachusetts Lowell
Email: {christopher_moralesgonzalez, matthew_harper}@student.uml.edu, xinwen_fu@uml.edu
† Email: yuanboya2022@outlook.com

***Abstract*—Building automation systems (BASs) provide convenience and comforts to building occupants, may optimize building management and energy consumption, and is part of the critical civil infrastructure—building. However, real-world attacks have been deployed against BASs. Our survey has identified software security is least studied. Our preliminary study has discovered tens of vulnerabilities in BAS devices and management software. This paper tries to gives a big picture of approaching BAS software security. We will focus on fuzzing, a natural way of identifying vulnerabilities of BAS device firmware and management software. We will introduce the fuzzing concepts and fuzzing tools. A case study of fuzzing a BACnet Secure Connect testbed is presented to demonstrate how to approach BAS fuzzing.**

I. Introduction

A building automation system carries out various building functions in an automated manner such as lighting systems, key card access systems, heating, ventilation, and air conditioning (HVAC). BAS is a type of cyber physical system (CPS) because it has cyber components that control these physical devices that interact with the physical environment. The architecture of a typical BAS has three levels: management (control center often with view of the entire network diagram), automation (scheduling) and field (physical devices such as sensors and actuators that interact with the environment). The global BAS market size was valued at $77.09 billion in 2023 and is projected to grow to $150.73 billion by 2032 [1].

Real-world attacks have been launched against BASs. In 2016 a DDoS attack using the Mirai botnet turned off the heating system for housing blocks in Finland [2]. In 2021 hackers locked out hundreds of building automation devices of a building automation engineering firm in Germany by setting a security key of those devices because those devices were connected to the Internet without protection [3]. In 2021, engineering computers in building automation systems in Pakistan, Afghanistan and Malaysia were infected with the ShadowPad backdoor [4]. In 2024, malware exploited the widely-used Modbus industrial communication protocol and caused over 600 apartment buildings in Ukraine to lose heat for two days in January 2024 [5]. It is said that this malware was the first industrial-controlled systems (ICS) malware that uses Modbus protocol to physically disrupt operational technology (OT).

We carried out a survey of BAS security [6] to gain an understanding of BAS security. We investigated BAS protocols such as BACnet [7], KNX [8], LonWorks [9], Modbus [10], ZigBee [11], Z-Wave [12], and EnOcean [13] while there are other BAS protocols such as ARCNET [14]. There are various attack surfaces against BASs including physical components, firmware/software, network and data. Conventional BAS protocols often have no security. Many BAS protocols now have dedicated security extensions, including BACnet Secure Connect , KNX Data Secure [15], KNX/IP Secure [16], ModBus/TCP Security [17], EnOcean High Security [18], ZigBee Pro [19] and Z-Wave Plus [20]. To secure BASs, we shall consider the following six factors: hardware, operating system (OS), software, networking, data and human, We make the following observations from our holistic survey: (i) network security has been extensively studied. However, much more is demanded. For example, there is lack of security for non-IP BAS networks such as BACnet/MSTP, which is broadly used and has no security extension. (ii) Hardware/OS security study is not well studied in the BAS domain. For example, secure bootstrapping shall be considered for BAS devices [21]. (iii) Software security is pretty much missing.

Among all attack surfaces against BASs, we haven't seen anything significant done for software security. Here the software refers to code in device firmware or BAS management software. Fuzzing is the under-explored domain for BAS software security. Fuzzing is a technique that involves passing random or unexpected inputs into an application to cause errors, unexpected behavior and even crashes. The challenges that come from fuzzing BASs are what's needed by fuzzing BASs and how a fuzzer, which injects random inputs into the target software, interplays with building automation systems. We often want to measure the effectiveness of a fuzz test. One critical metric is code coverage meaning how much code is actually hit during the test.

We have performed preliminary study of BAS software security through fuzzing [22]. We always assume the interface of the BAS software such as port number of a BAS server is known. We learn BAS message fields such as magic bytes with fixed values, length fields, counter fields and predictable passive fields and exclude them from fuzzing for efficiency since target BAS software would reject those fields with invalid values. Excluding those specific fields, we fuzz BACnet and KNX devices and software. 13 new vulnerabilities were discovered in BAcnet and KNX devices and software.

Given the promising results of our preliminary study of BAS software security, we want to share our thoughts of efficiently and effectively fuzzing BAS software and identifying root

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

causes of BAS software vulnerabilities.

II. Introduction to Fuzzing

In this section, we first introduce fuzzing in the overview. We then discuss types of fuzzing, and fuzzing effectiveness metric–code coverage.

A. Overview

Fuzz testing was originally developed as a software security testing technique by Barton Miller at University of Wisconsin in 1989 [23], [24]. To test a target device or software, a program called fuzzer is written, and it generates random data methodologically. The random data is injected into the target device or software through its interface such as network protocols, command-line arguments, file input and standard input. The application shall be closely monitored for any abnormalities such as software or system crashes, code execution errors and unexpected behaviors.

B. Fuzzing Workflow

A typical fuzzing test works as follows [25].

1) *Identify the target*. When fuzzing a BAS, the target could be a specific device located at a defined BAS address, such as a BACnet MS/TP MAC address.
2) *Determine the target input interface.* For a BAS fuzzing scenario, the input interface may be the BACnet network protocol.
3) *Generate fuzzy data.* The effectiveness and efficiency of fuzzing depend on the quality of generated test cases. It is essential to avoid test data that the target will instantly discard, as this would be inefficient. For instance, fuzzing may focus on modifying specific fields within a BACnet message.
4) *Inject fuzzy data into the target.* A fuzzer is responsible for feeding the generated test data into the target systemâĂŹs input interface to observe potential failures.
5) *Monitor the target.* Throughout the fuzzing process, the target must be observed for unexpected behaviors, such as crashes or anomalies.
6) *Analyze results.* If an issue arises, it is necessary to determine which test data triggered the problem. Additional analysis should assess the nature of the vulnerability and its exploitability. For example, does the issue stem from a buffer overflow? If so, can it be exploited, and how?

C. Types of Fuzzing

Fuzz testing can be categorized based on input generation methods into random fuzz testing, mutation-based fuzz testing, generation-based fuzz testing, and protocol-based fuzz testing.

1) Random fuzz testing: In random fuzz testing, the fuzzer produces completely random inputs and supplies them to the target. Although easy to implement, random fuzzers are often inefficient because the target may discard inputs that do not conform to the expected format or range.

2) Mutation-Based fuzz testing: In mutation-based fuzzing, a sample of data (often called corpse), such as a network traffic dump, is first collected. The fuzzer then systematically alters this data to generate new test cases.

3) Generation-based fuzz testing: In generation-based fuzzing, test cases are created according to a specified model or definition. This type of fuzzer typically requires in-depth knowledge of the target, such as its input format. While developing a generation-based fuzzer can be time-consuming, it tends to be more efficient and effective, as the test cases are less likely to be immediately rejected by the target.

4) Protocol-based fuzz testing: Protocol-based fuzz testing is a specific type of generation-based fuzz testing. Creating a protocol-based fuzzer demands a thorough understanding of the protocol under test. Protocol messages often have a specific sequence, where subsequent messages can only be sent after receiving a response to a previous one. A protocol-based fuzzer enables an in-depth examination of both the target and its protocol.

D. Code Coverage for Fuzzing Effectiveness

Evaluating the effectiveness and efficiency of a fuzzer can be challenging. Mutation-based fuzzers are capable of generating an unlimited number of test cases, which raises the question of when the fuzzer should stop. On the other hand, generation-based fuzzers create a finite set of test cases, but are those cases sufficient?

Code coverage is a metric used to measure how much of the code has been executed. There are different types of code coverage: Line or block coverage determines how many lines of code have been run; Branch coverage assesses how many branches, like conditional jumps, have been followed; and Path coverage looks at how many distinct paths through the code have been taken.

III. Fuzzing Tools

In this section, we first introduce the three functionalities of a fuzzing tool and then introduce four fuzzing tools that can be used BAS fuzzing: boofuzz, AFL++, AFLNet and WinAFL.

A. Functionalities of Fuzzing Tool

A fuzzing tool may implement three functionalities: test input generation, test input injection and bug detection [25]. The fuzzing workflow shall be automated.

Test input generation: There are various ways of generating fuzzing data as inputs to the target. There are quite a few fuzzing frameworks such as AFL++ [26], AFLNET [27], boofuzz [28] (successor of Sulley [29]). With a fuzzing framework, a programmer may follow the specification to write the fuzzing code while the framework provides fuzzing data.

Test input injection: Depending on the target's interface, the fuzzy data can be injected into the target accordingly. For example, we can convert an existing program/client as the fuzzer and inject the fuzzy data at appropriate point. With a fuzzing framework, a programmer may follow the specification to write a fuzzer while the framework provides fuzzy data as demonstrated in Fig. 1.

Bug detection: When the target program or device under fuzzing demonstrates issues, it may have bugs. For example, the program may crash. There are different types of crashes, such as segment faults and assert fails. We may run the target program under a dynamic memory error detector to discover potential memory errors related to dynamically allocated memory. We can also develop our own checker with Valgrind [30].

B. boofuzz

Boofuzz [31] is a network protocol fuzzing framework, and is a fork of Sulley [32], which is obsolete. Another related network protocol fuzzing framework is SPIKE [33], which is no longer under active development. Boofuzz provides APIs and allows the definition of a protocol, specifying which fields to fuzz. Therefore a boofuzz-based fuzzer is a generation-based fuzzer or protocol-based fuzzer. Boofuzz provides three main monitors including ProcessMonitor, NetworkMonitor and CallbackMonitor for monitoring failures such as crashes. Boofuzz provides logging mechanisms and saves log data of a test to a SQLite database. The log data can be viewed through a web interface.

Fig. 1 gives an example boofuzz script, i.e., a fuzzer. It has three parts. (i) Connect to target; (ii) Construct a fuzzing message template, which contains unfuzzable fields and one field to be fuzzed. Those fields often correspond to different fields of the application protocol; (iii) Fuzz. The fuzzer generate messages based on the message template, mutating the fuzzable field.

```
1.  host = '10.0.0.15'                                                    Connect to target
2.  port = 9999
3.  session=Session(sleep_time=1,
4.                  target=Target(connection=TCPSocketConnection(host, int(port))),
5.                  reuse_target_connection=True
6.  )
7.
8.  s_initialize("TRUN") # Initialize a new block request                 Const. fuzzing msg
9.  s_string('TRUN', fuzzable=False, name='TRUN-Command') # Push a string onto the block
10. s_delim(' .', fuzzable=False, name='TRUN-Space') # Push a delimiter onto the block
11. s_string('A', name='TRUN-STRING') # 'A' will be fuzzed
12. s_static('\r\n', name='TRUN-CRLF') # Push a static value onto the block stack; not fuzzed
13. session.connect(s_get("TRUN")) # s_get returns the request with the specified name   Fuzz
14. session.fuzz()
```

Fig. 1. Example Boofuzz Script

C. AFL++

AFL++ [26] is the most popular generic greybox fuzzing framework. It is developed based on AFL [34] and integrates many community improvements. AFL++ requires the source code or binary executables of target programs and adopts target-specific methods (e.g., source instrumentation, dynamic binary instrumentation, and hardware-based tracing) to collect coverage. As a generic fuzzer, AFL++ treats inputs as binary streams and mutates them at the byte level. Therefore, AFL++ works best for targets with unstructured input formats. However, it also provides interfaces to implement different mutation mechanisms and can be extended to handle complex input formats. By default, AFL++ injects (mutated) test cases through files and `stdin`, or directly writes them into the target memory. To support network-based fuzzing, AFL++ provides helper programs to inject inputs through network interfaces.

D. AFLNet

AFLNet [27] is a greybox network protocol fuzzer guided by both the coverage and execution status of the protocol. AFLNet assumes that the protocol packets contain status codes to represent the current protocol state. Therefore, AFLNet requires its users to extract protocol status code from protocol packets. With this, AFLNet monitors the states and state transitions encountered during fuzzing, and saves test cases triggering new protocol states or state transitions. The mutator of AFLNet operates at the packet level, but it ignores protocol formats and mutates each packet as if it is a plain binary stream.

E. WinAFL

AFL only supports fuzzing programs in Unix-like platforms and does not support Windows. To bridge this gap, WinAFL [35] is proposed to fuzz Windows binaries. Since Windows does not support forking a process, WinAFL adopts the persistent fuzzing (or, in-memory fuzzing) strategy, where it starts the target program, waits for it to enter a user-defined target function, and fuzzes the function for a specific number of iterations before it kills the process. WinAFL supports coverage collection via dynamic binary instrumentation or hardware-based tracing. WinAFL also comes with builtin support for network-based fuzzing.

IV. Case Study: Fuzzing BACnet

In this section, we present a case study—fuzzing a BACnet Secure Connect testbed. We first introduce the testbed and then present our findings.

A. Testbed

Fig. 2 illustrates our BACnet Secure Connect testbed, provided by Automated Logic. This testbed consists of the following devices: *(i) G5RE Router #1:* The G5RE router (top-left device of the panel) forwards BACnet packets between the BACnet Secure Connect network and the BACnet/IP network. *(ii) G5RE Router #2:* The second G5RE router (below G5RE #1) is configured to forward BACnet packets between the BACnet/IP network and the BACnet MS/TP network. *(iii) ZN551 Zone Controller:* This zone controller (bottom-left of the panel) is responsible for the three field devices in the bottom right of the panel. These are the Belimo CMB24-SR Damper Actuator (orange device), the Z2SP-ALC Temperature Sensor (below the actuator) developed by Automated Logic and the CX-100 Ceiling Occupancy Sensor (to the right of the actuator) created by Wattstopper. Each of these devices is able to be controlled and interacted with via exposed BACnet objects on the ZN551 controller. *(iv) BACnet/IP Controller:* The Optiflex IP Controller (top-right of the panel) developed by Automated Logic is similar to the ZN551 Zone Controller. However, it facilitates the communications between the devices under its

control and the overarching BACnet/IP network. *(v) BACnet/IP-to-BACnet/IP Router:* The BACnet/IP-to-BACnet/IP Router (middle-right of the panel) developed by Cimetrics is used to route communications between two BACnet/IP networks.

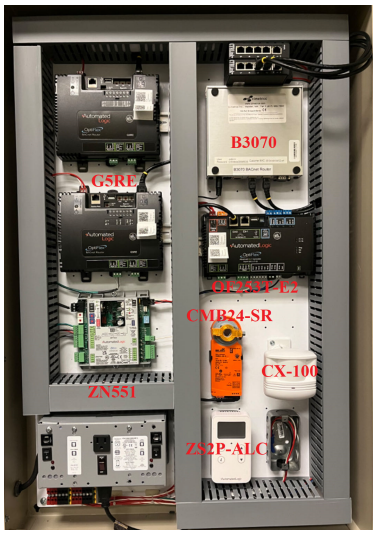

Fig. 2. BACnet Secure Connect Testbed

The panel can be managed by the following BAS management software. *WebCtrl:* Not shown in the figure is the Building Management System (BMS) WebCtrl developed by Automated Logic. This is hosted on a Windows 10 Desktop connected to the Ethernet switch above the BIP-to-BIP router. This is the central management platform for the entire panel, which allows us to interact with all devices and datapoints for the devices in the panel. *Secure Connect Hub* Also not shown in the figure is the Secure Connect Hub developed by Automated Logic. This is also hosted on a Windows 10 Desktop. This component is required for all BACnet/SC communications in the panel. This central hub will receive communications from the other devices in the Secure Connect network and will forward the message to the other devices connected to the hub.

The panel has three forms of BACnet communications: BACnet MS/TP, BACnet/IP and BACnet/SC. In the BACnet/SC network, we have WebCtrl, the Secure Connect Hub and the G5RE #1. In the BACnet/IP network, we have the G5RE #2, the BACnet/IP-to-BACnet/IP router, and the IP controller. In the MS/TP network lies under the ZN551 Zone Controller. The devices themselves are connected to the controller through its RNET port (a proprietary communication protocol).

A typical communication flow can be shown between the end devices (such as the Temperature Sensor) up to WebCtrl. For instance, once the sensor wants to report the temperature, it will send its message to the ZN551 controller, which then converts this to a BACnet MS/TP packet and sends it to G5RE #2 through the MS/TP port on the device. This router then strips the MS/TP header of the packet and converts it to a BACnet/IP packet aimed at G5RE #1. Next, this router will convert this insecure BACnet/IP packet and changes it to a BACnet Secure Connect packet and will forward it to the Secure Connect Hub. Finally, the Secure Connect Hub forwards this packet to WebCtrl.

The devices utilizing BACnet Secure Connect will utilize digital certificates signed by a custom Certificate Authority (CA) created specifically for this panel. Any communications on the BACnet Secure Connect network will require these certificates and corresponding encryption keys as they will utilize TLS-enabled Websockets to encapsulate and secure BACnet packets.

B. Findings

We conducted fuzz testing on various devices in our panel and identified two major bugs affecting a total of four devices. In adherence to responsible disclosure practices, both bugs have been reported to Automated Logic. At the companyâĂŹs request, we cannot disclose specific details about the vulnerabilities at this time, as a fix has not yet been released. However, we provide general insights and discuss the broader implications of these issues.

1) Fuzzer Design: We used the Yabe library as the basis for our random fuzzer, as it provides native support to communicate through all communication mediums on our panel. For example, the two bugs discussed next will abuse the inherent communication mechanisms within the panel facilitated by the various routers and the Secure Connect Hub, as well as vulnerabilities inherent within the BACnet protocol (more specifically, the lack of authentication within insecure BACnet networks). Our fuzzer will connect to the MS/TP bus on the panel through the ZN551 Zone Controller's MS/TP port. It will carefully craft the packets to target a specific device. When attacking a device in the Secure Connect network, it will maintain the Data Link and Network layers of the packets to allow it to be processed and forwarded through these networks - the Application layer remains untouched, allowing us to send fuzzed data to these devices.

The BACnet specification requires a specific *Device Object* to be present and available at all times on the BACnet devices in the network. This *Device Object* will hold information about the device such as the device's name, BACnet instance number, serial number, manufacturer, and any BACnet objects that the device has. This object must remain present at all times for other devices to discover and interact with the device. This Device Object is what our fuzzer uses to understand if the target device is in some fault state. To account for network latency or network inconsistencies (i.e. dropped packets), we perform three sequential health checks, which query this Device Object to confirm the fault state.

2) Discovered Bugs: The first bug discovered affects WebCtrl, the Secure Connect Hub, and the G5RE #1. Our fuzzer had targeted the Application layer of BACnet packets and had caused an *Array-Index-Out-of-Bounds* error on the device. Overtime, our fuzzer had generated over 100+ of these messages in which the devices had become unresponsive to the fuzzer's health check (e.g. querying the device for this Device Object). After gathering the device's system logs, it became clear that if we sent more than 100 packets that caused this

error, the Device Object process within the device would not restart - resulting in the Device Object becoming unavailable. This Denial-of-Service (DoS) has implications on the BACnet network in the case if a device requires this object to perform any complex automation. If this object is not available, this would not be possible, potentially having severe consequences. Each device requires a full restart of the program or the device to restore functionality, resulting in a temporary DoS.

The second bug affects the ZN551 Zone Controller. Again, the fuzzer was targeting the Application Layer of BACnet messages aimed at this device. The fuzzer had crafted a message in which it seems (awaiting a response from the company) an *Array-Index-of-Bounds* error was not handled properly. After the fuzzer had reported a fault state, a visual inspection of the device was performed. The result of this crash had led to the LED on the device flashing red indicating an error. Upon logging into WebCtrl to gather a status report of the device, the controller was completely unresponsive to these requests and did not forward the values of the temperature sensor, actuator, or occupancy sensor. This device requires a full power cycle to restore normal, full functionality. This can pose serious risks in the case of a fire alarm system. If this zone controller has sensors or water sprinklers and this attack is carried out, this system may not activate in time.

V. Conclusion

In this paper, we share our thoughts of using fuzzing to study software security of BASs. For fuzzing BAS devices with knowledge of device firmware, boofuzz or customized fuzzing tools can be good choices. For fuzzing BAS management software without source code, AFL++, AFLNet or WinAFL can be a good choice depending on what operating system the software runs on. There is a lack of tools on fuzzing Java or C# based BAS management software running on Windows. BAS software security is still an open field and requires significant efforts. Our future work includes development of various fuzzers to test the BAS device and management software security and identify the root causes of the vulnerabilities.

References

[1] Fortune Business Insights Pvt. Ltd., "Building automation systems market size," https://www.fortunebusinessinsights.com/building-automation-systems-market-107861, 2024, accessed: 2024-12-9.

[2] L. Mathews, "Hackers use ddos attack to cut heat to apartments," "https://www.forbes.com/sites/leemathews/2016/11/07/ddos-attack-leaves-finnish-apartments-without-heat/?sh=7cfc8dfa1a09", Nov 2016.

[3] Veridify Security Inc., "Cyberattacks shut down building automation systems," https://www.veridify.com/cyberattacks-shut-down-building-automation-systems/, December 2021, accessed: 2024-12-9.

[4] AO Kaspersky Lab, "Attacks on industrial control systems using shadowpad," https://ics-cert.kaspersky.com/publications/reports/2022/06/27/attacks-on-industrial-control-systems-using-shadowpad/, June 2022, accessed: 2024-12-9.

[5] C. Vasquez, "Simple âĂŸfrostygoopâĂŹ malware responsible for turning off ukrainiansâĂŹ heat in january attack," https://cyberscoop.com/frostygoop-ics-malware-dragos-ukraine/, July 2024, accessed: 2024-12-9.

[6] C. Morales-Gonzalez, M. Harper, M. Cash, L. Luo, Z. Ling, Q. Z. Sun, and X. Fu, "On building automation system security," *High-Confidence Computing*, vol. 4, no. 3, p. 100236, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295224000394

[7] A. S. Publications, "Ansi/ashrae standard 135-2020," ASHRAE, Tech. Rep., 2020.

[8] *KNX Standard Specifications*, v2.1 ed., KNX, 2013.

[9] *Introduction to the LonWorks System*, 1st ed., Echelon Corporation, 1999.

[10] *MODBUS APPLICATION PROTOCOL SPECIFICATION*, v1.1b3 ed., Modbus Organization, 2012.

[11] *ZigBee Specification*, 05th ed., ZigBee, 2015. [Online]. Available: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

[12] *Z-Wave Device Class Specification*, Z-Wave Alliance, Beaverton, United States, 3 2021.

[13] E. S. IoT, "Radio technology," https://www.enocean.com/en/technology/radio-technology/#:~:text=The%20standard%20uses%20the%20868,and%20928%20MHz%20in%20Japan., 2021.

[14] "Arcnet âĂŤ an embedded real-time network," https://arcnet.cc/abtarc.htm, accessed: 2024-12-10.

[15] *Application Note 158/13 v02 KNX Data Security*, 2nd ed., KNX, 2013.

[16] *Application Note 159/13 v04 KNXnet/IP Secure*, 2nd ed., KNX, 2013.

[17] *MODBUS/TCP Security*, v21 ed., MODBUS, 2018.

[18] *Security of EnOcean Radio Networks*, v2.5 ed., EnOcean Alliance, 2018.

[19] C. S. Alliance, *ZigBee Pro Specification*, 05th ed., Connectivity Standard Alliance, 2023.

[20] *Z-Wave Plus v2 Device Type Specification*, Z-Wave Alliance, Beaverton, United States, 10 2021.

[21] B. Parno, J. M. McCune, and A. Perrig, "Bootstrapping trust in commodity computers," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 414–429.

[22] Y. Zhang, Z. Ling, M. Cash, Q. Zhang, C. Morales-Gonzalez, Q. Z. Sun, and X. Fu, "Collapse like a house of cards: Hacking building automation system through fuzzing," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1761âĂŞ1775. [Online]. Available: https://doi.org/10.1145/3658644.3690216

[23] B. P. Miller, "Fuzz testing of application reliability," https://pages.cs.wisc.edu/~bart/fuzz/fuzz.html, accessed: Nov. 10, 2024.

[24] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, no. 12, p. 32âĂŞ44, Dec. 1990. [Online]. Available: https://doi.org/10.1145/96267.96279

[25] "Find software bugs," https://www.cs.ucf.edu/~czou/CAP6135-16/findingBug.ppt, accessed: 2024-12-10.

[26] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "AFL++: Combining incremental steps of fuzzing research," in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association, Aug. 2020.

[27] V.-T. Pham, M. BĂűhme, and A. Roychoudhury, "Aflnet: A greybox fuzzer for network protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 2020, pp. 460–465.

[28] "boofuzz: Network protocol fuzzing for humans," https://boofuzz.readthedocs.io/en/stable/, accessed: Nov. 13, 2024.

[29] "Sulley," https://github.com/OpenRCE/sulley, accessed: Nov. 13, 2024.

[30] "Valgrind," https://valgrind.org/, accessed: 2024-12-10.

[31] J. Pereyda, "Boofuzz: Network protocol fuzzing for humans," https://github.com/jtpereyda/boofuzz, 2016, accessed: 2024-12-5.

[32] P. Amini, "Sulley," https://github.com/levigross/Sully, 2007, accessed: 2024-12-16.

[33] G. M. Ferreira, "SPIKE," https://github.com/guilhermeferreira/spikepp, 2017, accessed: 2024-12-17.

[34] M. Zalewski, "American fuzzy lop," 2018. [Online]. Available: https://lcamtuf.coredump.cx/afl/

[35] I. Fratric, "Winafl," 2016. [Online]. Available: https://github.com/googleprojectzero/winafl