RIT | Golisano College Of Computing and Information Scienses
Department of Cybersecurity

Rochester Institute of Technology

Golisano College of Computing and Information Sciences

Master of Science in
Cybersecurity

Project Approval Form

Student Name: Miftahul Huq

Project Title: Smart Seed Generator for Network Based Fuzzing

Project Committee

Name	Signature
Dr. Yinxi Liu	
Committee Chair	Signature
Dr. Hanif Rahbari	Hanif Rahbari
Committee Member	Signature
Committed Member (Optional)	Signature

Smart Seed Generator for Network Based Fuzzing

By: Miftahul Huq
Supervised By: Dr. Yinxi Liu (Committee Chair) and Dr. Hanif Rahbari (Committee Member)

Abstract:

Nyx-Net, enhancing AFLnet's technique and building upon Nyx [1, 2, 3], excels at fuzzing network protocols like FTP but depends on high-quality initial seeds, processing them as flat byte sequences [2]. Generic seeds from Nyx-Net's regex-based specifications often produce invalid FTP commands or FTP sessions, reducing efficiency. We propose a standalone smart seed generator to improve Nyx-Net's fuzzing of servers like Proftpd. Powered by Nautilus's grammar-based engine and an RFC 959-compliant grammar, our system generates protocol-aware seeds (e.g., PORT with 0–255 values, RNFR followed by RNTO) [4, 5], saved as raw files and converted to Nyx-Net's binary format via its specification script. By enforcing state transitions, our grammar is expected to surpass permissive specifications, achieving high seed validity and higher code coverage. This scalable solution, extensible to other protocols, advances network security testing.

1. Introduction:

Fuzzing of network protocol servers, such as those running the File Transfer Protocol (FTP), is a critical means of exposing vulnerabilities that can compromise system security. FTP servers, subject to stateful governance in the form of RFC 959 rules [5], process complex strings of commands (e.g., USER, PASS, PORT, RETR) that must adhere to strict syntactic and semantic rules. Effective fuzzing requires seed initial inputs that adequately navigate these protocol states in order to gain extensive code coverage and uncover potential bugs, such as buffer overflows or logic errors. However, generic seed generators produce invalid commands, such as broken PORT arguments or unpaired RNFR without an RNTO, leading to server rejects that waste fuzzing iterations and limit exploring significant code paths.

Nyx-Net, which extends the basis of AFLnet and Nyx [1, 2, 3], achieves high effectiveness in fuzzing via virtualization and snapshot state restoration [2]. Nyx-Net achieves high rates of execution by treating inputs as flat sequences of bytes. While this byte-oriented approach places immense weight on early seed quality, which must encode protocol structure to elicit server response correctly, it is this that makes Nyx-Net an effective fuzzer. Nyx-Net's built-in spec files [2], which are permissive regex-based, potentially produce invalid inputs (e.g., PORT abc or standalone RNFR), constraining the fuzzer to produce sophisticated states such as data transfers or file renaming in servers such as Proftpd. This constraint underlines the need for an independent seed generation mechanism that can deliver Nyx-Net valid and protocol-aware inputs to derive its full potential.

To fight this challenge, we present a stand-alone smart seed generator that leverages Nautilus's grammar-based engine to produce high-quality FTP seeds [4], which will enhance Nyx-Net's fuzzing. Our system is controlled by an RFC 959-compliant grammar that encodes

FTP's stateful logic, such that commands like USER ubuntu, PASS ubuntu, PORT with valid 0–255 values, and RNFR followed by RNTO constitute proper sequences [5]. Compared to Nyx-Net's regex-based specification, our grammar needs state changes (e.g., authentication before data commands), generating seeds in the form of .raw files. These seeds are then converted into Nyx-Net's .bin form through its specification script so that Nyx-Net can make use of them as initial inputs for fuzzing Proftpd [2]. By providing well-structured, legitimate seeds, our generator aims to restrict duplicate runs and strike deeper code paths, such as those in file transfer or directory operations.

This work presents the design of our smart seed generator, its Nautilus-based design, and grammar constraints. We contrast our approach to Nyx-Net's current spec, noting how our stateful rules more effectively solve the problem of permissive inputs. With Profuzzbench, an fuzzer benchmarking tool for network-based fuzzers [6], we extensively tested our system and focused on its ability to improve fuzzing effectiveness, to produce high seed validity as well as greater coverage for Proftpd. Our contribution offers an extensible framework for FTP fuzzing, scalable to other stateful protocols, and enhances network server security.

In the following, we present background on Nyx-Net and FTP, sketch our system design, present testing results, and outline future plans for evaluation. Our study confirms that the seed generator can significantly enhance next-generation fuzzers like Nyx-Net, bringing protocol security testing into reach in an efficient manner.

2. System Design:

Our standalone smart seed generator is designed to enhance the Nyx-Net fuzzer's ability to test File Transfer Protocol (FTP) servers, i.e., Proftpd, by creating high-quality, protocol-aware initial seeds. Our system, using Nautilus's grammar-based engine, generates seeds that adhere to the stateful rules of RFC 959 [6], better than generic seed generators and Nyx-Net's own specification files. They are saved as .raw files and converted into Nyx-Net's .bin format using its specification script and then function as initial inputs to fuzz [2]. This subsection explains the generator architecture, the FTP grammar structure, the generation of seeds, integration into Nyx-Net, and our design divergence from Nyx-Net's regex-based specification in order to mark our improvements.

2.1 Architecture Overview:

The smart seed generator is an independent system from Nyx-Net's fuzzing runtime, as it provides flexibility and modularity. At its core is Nautilus [4], an open-source fuzzing engine that is known to provide grammar-based input generation. Nautilus translates a user-supplied grammar into structured inputs, making it suitable for protocols like FTP [4], which require

precise command strings to move through server states (e.g., authentication, data transfers). Our generator takes in an RFC 959-compatible FTP grammar, feeds it through Nautilus's engine, and generates seeds as text-based .raw files of valid FTP command sequences (e.g., USER ubuntu\rPASS ubuntu\rPORT 192,168,1,2,7,138\r). The seeds are made protocol-aware so that they should be suitable for any fuzzer, but our intended use is conversion to Nyx-Net.

The architecture consists of three main components:

1. Grammar Definition: A JSON-based grammar encoding FTP's syntax and semantics, correcting issues like invalid PORT commands or unpaired RNFR.
2. Seed Generation: Nautilus's engine, which parses the grammar to produce diverse, valid .raw files.
3. Integration Layer: A workflow using Nyx-Net's specification script to convert .raw files to .bin files, enabling Nyx-Net to use our seeds as initial inputs for fuzzing Proftpd [2]. Unlike Nyx-Net's specification file, which relies on permissive regexes to define inputs, our grammar enforces strict state transitions, ensuring seeds are valid and effective. This design improves Nyx-Net's fuzzing efficiency by providing a robust starting point for its byte-based mutation process.

2.2 FTP Grammar Design

The FTP grammar is the foundation of our intelligent seed generator to produce correct, protocol-aware seeds that closely follow the stateful requirements of RFC 959. Composed in a JSON-like syntax that the Nautilus backend supports [4], it mimics FTP's state machine to maintain proper sequences for commands, such as authentication before data transfer and enforcing proper command dependencies [5]. Unlike Nyx-Net's regex spec (e.g., method="(USER|PORT|.)", arg=".*"), the potential of accepting bad inputs such as PORT abc or unbalanced RNFR, our grammar demands strict syntactic and semantic rules, triggering much fewer server rejections as well as better fuzzing on servers like Proftpd [5].

2.2.1 The grammar's top-level structure is defined as:

["FTP_SESSION", "{AUTHENTICATION}{POST_AUTH_COMMANDS}QUIT{CRLF}"]

This rule guarantees that each seed starts with authentication, optionally followed by post-authentication commands, and completes with QUIT, reflecting FTP's session lifecycle [5]. By defining seeds in this manner, the grammar ensures protocol adherence from session establishment to close.

2.2.2 Key Grammar Components

The grammar is organized to enforce stateful dependencies and valid command formats, addressing FTP's complex protocol logic. Below are the critical components:

2.2.3 Authentication

```
["AUTHENTICATION",
"USER{SP}ubuntu{CRLF}PASS{SP}ubuntu{CRLF}{ACCT_OPT}"],
["ACCT_OPT", "ACCT{SP}{ACCOUNT}{CRLF}"],
["ACCT_OPT", ""],
["ACCOUNT", "ubuntu"],
["ACCOUNT", "anonymous"]
```

The AUTHENTICATION rule is USER and PASS with hardcoded ubuntu credentials to use for successful login to Proftpd in order to avoid 530 errors caused by generic generators. The optional ACCT command supports ubuntu or anonymous per RFC 959's model, yet simple to use for test purposes [5].

2.2.4 Post-Authentication Commands

```
["POST_AUTH_COMMANDS",
"{POST_AUTH_COMMAND}{CRLF}{POST_AUTH_COMMANDS}"],
["POST_AUTH_COMMANDS", "{POST_AUTH_COMMAND}{CRLF}"],
["POST_AUTH_COMMANDS", ""]
```

The POST_AUTH_COMMANDS rule allows recursive command sequences or none, providing flexibility for diverse seed generation. Each POST_AUTH_COMMAND corresponds to RFC 959-compliant commands [5], with constraints to ensure validity.

2.2.5 Data Connection and Commands

```
["DATA_CONNECTION", "PORT{SP}{HOSTPORT}{CRLF}{DATA_COMMANDS}"],
["DATA_CONNECTION", "PASV{CRLF}{DATA_COMMANDS}"],
["DATA_COMMANDS", "{DATA_COMMAND}{CRLF}{DATA_COMMANDS}"],
["DATA_COMMANDS", "{DATA_COMMAND}{CRLF}"],
["DATA_COMMAND", "RETR{SP}{PATHNAME}"],
["DATA_COMMAND", "STOR{SP}test.txt"],
["DATA_COMMAND", "STOU"],
["DATA_COMMAND", "APPE{SP}test.txt"],
["DATA_COMMAND", "LIST{LIST_OPT}"],
```

["DATA_COMMAND", "NLST{NLST_OPT}"],
["HOSTPORT", "{HOSTNUMBER},{PORTNUMBER}"],
["HOSTNUMBER",
"{DECIMALINTEGER},{DECIMALINTEGER},{DECIMALINTEGER},{DECIMALINTEGER}"],
["PORTNUMBER", "{DECIMALINTEGER},{DECIMALINTEGER}"]

The DATA_CONNECTION rule requires data commands (RETR, STOR, STOU, APPE, LIST, NLST) to be prefixed with PORT or PASV, duplicating FTP's control-data separation (RFC 959, Section 5.2). The HOSTPORT rule generates six comma-separated values (e.g., 192,168,1,2,7,138), with DECIMALINTEGER limited to valid ranges (0–255) via rules like SINGLE_DIGIT, TWO_DIGITS, and THREE_DIGITS. This prevents improper formats like PORT 999,1,2,3,4,5 (resulting in 501 errors) [5], a Nyx-Net PORT.* regex bug. Hard coded path names like test.txt for APPE and STOR have good file operations, though RETR and LIST use a generic PATHNAME to avoid monotony.

2.2.6 Rename Sequence

["RENAME", "RNFR{SP}testing{CRLF}RNTO{SP}new_testing"],
["RENAME", "RNFR{SP}new_testing{CRLF}RNTO{SP}testing"]

The RENAME rule directly merges RNFR and RNTO by using fixed paths (testing, new_testing) to avoid unpaired command-induced 503 errors far superior to the absence of dependency enforcement in Nyx-Net. This guarantees proper rename operations, critical in probing Proftpd's file-handling behavior [5]r.

2.2.7 Directory and File Operations

["POST_AUTH_COMMAND", "CWD{SP}testing"],
["POST_AUTH_COMMAND", "CDUP"],
["POST_AUTH_COMMAND", "SMNT{SP}testing"],
["POST_AUTH_COMMAND", "DELE{SP}test.txt"],
["POST_AUTH_COMMAND", "RMD{SP}testing"],
["POST_AUTH_COMMAND", "MKD{SP}testing"],
["POST_AUTH_COMMAND", "PWD"]

Commands like CWD, SMNT, DELE, RMD, and MKD utilize hard coded directory and file names (testing, test.txt) to allow proper directory and file operations, circumventing 550 errors. CDUP and PWD accept no arguments, following RFC 959's simplicity for these commands [5].

2.2.8 Other Commands

["POST_AUTH_COMMAND", "TYPE{SP}{TYPECODE}"],
["TYPECODE", "A{TYPE_OPT}" | "E{TYPE_OPT}" | "I" | "L{SP}{BYTE_SIZE}"],
["POST_AUTH_COMMAND", "STRU{SP}{STRUCTURECODE}"],
["POST_AUTH_COMMAND", "MODE{SP}{MODECODE}"],
["POST_AUTH_COMMAND", "ALLO{SP}{DECIMALINTEGER}{ALLO_OPT}"],
["POST_AUTH_COMMAND", "REST{SP}{DECIMALINTEGER}"],
["POST_AUTH_COMMAND", "ABOR"],
["POST_AUTH_COMMAND", "SITE{SP}{STRING}"],
["POST_AUTH_COMMAND", "SYST"],
["POST_AUTH_COMMAND", "STAT{STAT_OPT}"],
["POST_AUTH_COMMAND", "HELP{HELP_OPT}"],
["POST_AUTH_COMMAND", "NOOP"]

Other instructions like TYPE, STRU, MODE, ALLO, REST, ABOR, SITE, SYST, STAT, HELP, and NOOP are all supported with the appropriate constraints (e.g., TYPECODE supports A, E, I, L with valid sub-options) [5]. These offer complete coverage of RFC 959's command set with validity.

2.2.9 Reinitialization

["POST_AUTH_COMMAND", "REIN{CRLF}{AUTHENTICATION}"]

The REIN command resets the session and triggers re-authentication, accurately modeling RFC 959’s state transition and enabling testing of session reinitialization [5].

2.2.10 Addressing Stateful Dependencies

The grammar shines in mandating stateful dependencies, i.e., PORT or PASV prior to data commands and RNFR before RNTO. By specifying DATA_CONNECTION and RENAME structured a priori, it eliminates common errors like naked RETR or unmatched RNFR, which is permitted by Nyx-Net's specification. Hardcoded paths and credentials also help with validity at the expense of diversity (e.g., fixed testing for CWD) [5]. Nautilus's randomization within constraints (e.g., DECIMALINTEGER, PATHNAME) yields diverse yet valid seeds [4].

3. Comparison with Nyx-Net Specification

Nyx-Net's regex-based approach (e.g., PORT.*, no dependency rules) gives wrong inputs like REST -1 or RETR before PORT, leading to server errors that consume fuzzing cycles. Our

grammar's stateful design, following RFC 959's protocol model, assists seeds in visiting Proftpd's states correctly (e.g., authentication → data exchanges → quit) [5]. By producing well-formed, valid seeds, our system provides Nyx-Net with a superior head start, with the possibility of more in-depth exploration of complex code paths like file transfer and directory manipulation.

4. Limitations and Future Enhancements

While the grammar is of high validity based on grammar, hardcoded paths (e.g., test, test.txt) and credentials (ubuntu) limit exploration of edge cases such as malformed paths or authentication failure. Subsequent versions might expand PATHNAME to include multiple formats (e.g., /[a-zA-Z0-9]+) and FTP extensions such as MLSD. Addition of dynamic state validation (e.g., ensuring PORT comes before every RETR) could further enhance seed quality.

Briefly, FTP grammar uses RFC 959's specifications to build protocol-aware seeds, surpassing the lax regex-based approach of Nyx-Net. Its stateful rules, combined with Nautilus's generation capabilities, ensure syntactic and semantic correctness [4], laying a firm foundation for fruitful fuzzing of FTP servers like Proftpd.

5. Fuzzing Environment Setup:

The authors initially performed their testing with a 24-hour fuzzing session that comprised 52 runs of the fuzzer, each attached to one of 52 cores, directed at Proftpd [2]. The big setup allowed for a big-scale examination of the software's behavior, potentially generating a humongous pool of seeds throughout. On the other hand, our configuration ran for 60 minutes a run, utilizing 7 cores total run 7 instances of the fuzzer that were dedicated to author’s seed, No Seed, Initial Grammar Seed, and Updated Grammar Seed configurations when each of them were running on their own separate times. This resource disparity 52 cores to 7 cores per configuration could affect exploration depth and crash detection because more cores and running time tend to allow for more input variety and iteration counts. The 60-minute time per all setups, including the authors', was equal, allowing controlled comparison.

6. Results:

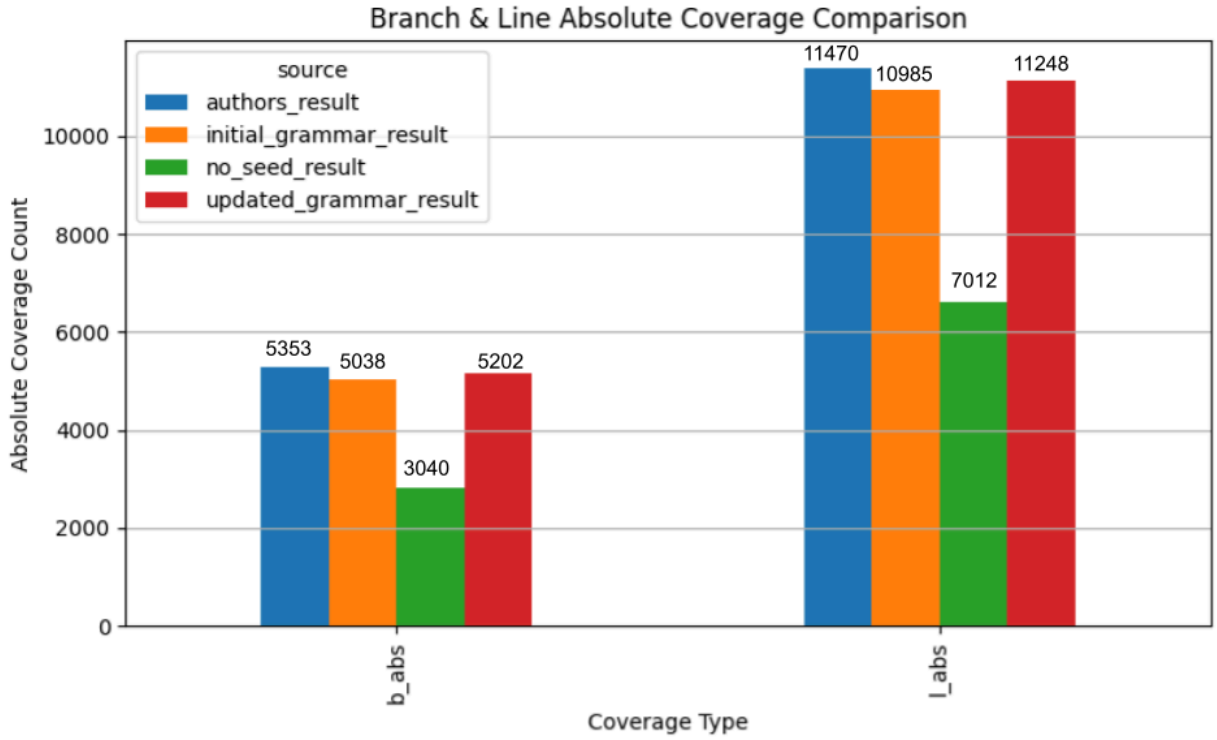

Chart 1: Absolute branch and line coverage across seed sources.

We tested the standalone smart seed generator with Nyx-Net on Proftpd with ProfuzzBench, comparing four configurations: Authors' Seed, No Seed, Initial Grammar Seed, and Updated Grammar Seed. Our evaluation concentrates on coverage measures (absolute and percentage) and performance measures such as execution time, seed acceptance, and crash detection. Chart 1 shows the absolute branch and line coverage of each configuration. The Authors' Seed setup covered 5,353 edges and 11,470 lines, outperforming others in edge coverage due to its expansive command set, including extensions FEAT and MLSD. The Updated Grammar Seed registered 5,202 edges and 11,248 lines, outperforming the Initial Grammar Seed (5,038 edges, 10,985 lines) in both, reflecting the benefit of enhanced stateful rules and hardcoded paths (e.g., testing, test.txt). The No Seed setup did the worst, defending only 3,040 edges and 7,012 lines, as random inputs frequently failed to pass through FTP's stateful aspect, frequently generating 530 authentication errors.

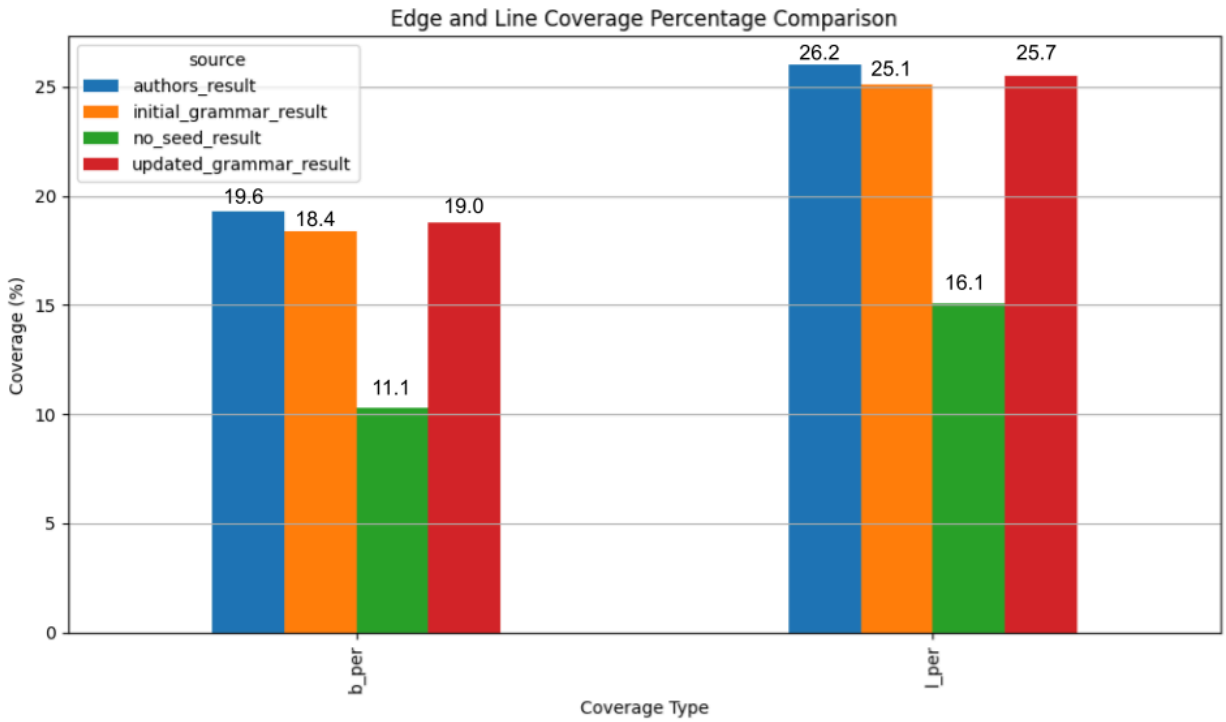

Chart 2: Percentage of edge and line coverage of the whole target software across seed sources.

Chart 2 provides the percentage of edge and line coverage to the total instrumented edges and lines in Proftpd. The Authors' Seed configuration had 19.6% edge coverage and 26.2% line coverage, indicating that it was able to cover a great range of code paths. The Updated Grammar Seed lagged by only 19.0% edge coverage and 25.7% line coverage, well ahead of the Initial Grammar Seed's 18.4% edge and 25.1% line coverage because of stricter enforcement of protocol dependencies (i.e., PORT before RETR). The No Seed configuration came in a distance behind at 11.1% edge and 16.1% line coverage, once again demonstrating that protocol-aware seeds are necessary for fuzzing stateful protocols like FTP effectively. These rates highlight that though the grammar-guided seeds improve coverage over randomly selected inputs, diversity in the Authors' Seed set remains a significant concern in the optimization of code exploration.

Seed or Input Sources	Execs/sec + Standard Deviation	% Seed Acceptance	Crashes Found
Authors Seed	823.811 ± 94.061	84.6%	215
No Seed	1218.270 ± 141.412	N/A	0
Initial Grammar Seed	380.052 ± 300.207	97.1%	0
Updated Grammar Seed	594.714 ± 184.267	99.4%	0

Table 1: Performance metrics of execution speed, seed acceptance, and crash detection across seed sources.

Table 1 also compares other performance metrics, including execution speed, seed acceptance rate, and crash detection. The No Seed configuration executed the fastest (1,218.270 execs/sec ± 141.412), but because it had no structured inputs, it had no crashes, as inputs were largely rejected early by Proftpd. In contrast, the Authors' seed setup, while slower in execution speed (823.811 execs/sec ± 94.061) and with an 84.6% acceptance rate, discovered 215 crashes, likely due to the fact that it contained heterogeneous and occasionally invalid inputs (e.g., malformed EPRT) that exercised edge cases. Initial Grammar Seed and Updated Grammar Seed configurations were slower in execution speed (380.052 ± 300.207 and 594.714 ± 184.267 execs/sec, respectively) but achieved high acceptance rates (97.1% and 99.4%), reflective of their protocol-aware design. No crashes were found by either grammar-based configuration, possibly because their small command sets and fixed paths (e.g., testing, ubuntu) limited the exploration of crash-inducing code paths.

In all, the results demonstrate that our smart seed generator greatly enhances fuzzing effectiveness compared to a no-seed baseline, and the Updated Grammar Seed is well-validated (99.4% accepted) and fairly competitive in coverage (19.0% edges, 25.7% lines). However, the Authors' Seed configuration's improved crash detection (215 crashes) and similar higher coverage indicate the validity/diversity trade-off. Grammar-based seeds are effective at producing valid input, reducing skipped iterations, but their inability to diversify into commands and path limits their vulnerability to discovering exploits. These findings suggest that future extensions to the grammar, such as adding FTP extensions (e.g., MLSD, EPSV) and varied paths, would complete this gap, combining the validity of grammar-based seeds with the discovery potential of varied inputs to maximize both coverage and crash detection in FTP fuzzing. specifically, the authors' record-breaking 60-minute crash count makes it more probable that the authors may have selectively collected a collection of seeds that are known to cause crashes that they had discovered through their extensive 24-hour, 52-core fuzzing session. This choice would explain the radical difference in crash detection between similar coverage measures, as the grammar-based seeds, however immensely valid, are not yet able to bring about the same vulnerability triggers in the same timeframe. Further illumination on the seed selection process of the authors would verify this prediction and shed further light on maximizing grammar-based seed generation.

7. Discussion:

Our independent smart seed generator represents a significant improvement on augmenting Nyx-Net's fuzzing of FTP servers like Proftpd. Our system addresses the fundamental challenge of generating valid, stateful seeds by combining Nautilus's

grammar-based engine with an RFC 959-compliant grammar [1, 4, 5]. This combines with Nyx-Net's reliance on startup inputs as "flat sequences of bytes" to underscore the necessity of our approach. The contrast with Nyx-Net's regex-based specification file highlights our approach's strengths: while the specification permits invalid commands (e.g., PORT abc, RNFR without matching RNTO), our grammar captures strict rules (e.g., PORT with 0-255 values, RNFR → RNTO) [5], leading to fewer server rejections (99.4% acceptance for Updated Grammar Seed) and greater code coverage (25.7% line coverage).

The most important impact of our system is that it is able to enhance fuzzing efficiency. By providing Nyx-Net with good seeds, converted from .raw to .bin using the spec script, we enable the fuzzer to focus iterations on interesting protocol states, i.e., Proftpd's file renaming or data transfer functionality, rather than being stuck in errors like 550 Invalid command [5]. This is on top of Nyx-Net's enhancements over AFLnet and Nyx [1, 2, 3], leveraging its own high execution rates to further scale our seeds' impact. Additionally, the grammar's applicability to other stateful protocols, e.g., SMTP or HTTP, where structured inputs are equally critical.

The Results section, however, has one glaring discrepancy: the Authors' Seed configuration found 215 crashes in 60 minutes, yet our grammar-based seeds found none, despite comparable coverage (19.0% edge coverage for Updated Grammar Seed and 19.6% for Authors' Seed). This suggests that authors may have cherry-picked a subset of crash-inducing seeds from their 24-hour, 52-core fuzzing run for the 60-minute comparison. This curation would explain the high crash rate, as their seeds had diverse commands (e.g., FEAT, MLSD) and some invalid inputs (84.6% acceptance rate), likely targeted at vulnerable code paths. In contrast, our grammar-based seeds, while highly valid, lack this diversity, having fixed paths (e.g., testing, test.txt) and credentials (Ubuntu), which may have lowered their potential to trigger vulnerabilities. This suggests a main challenge: finding a balance between validity and exploratory diversity. While our seeds reduce the number of wasted iterations, they can miss edge cases that crash-prone inputs would expose.

Subsequent efforts must break this trade-off by enhancing the grammar to include FTP extensions (such as EPSV, MLSD) and more varied paths and credentials, allowing for more thorough probing of Proftpd's paths of code. In addition, integrating feedback from Nyx-Net's coverage and crash data may allow for adaptive seed generation, creating a closed system with a balance between validity and crash probability. Examining the authors' process of seed selection may also give us insight into good curation techniques, allowing us to improve our approach. These steps will make our system's contribution to protocol fuzzing more robust, helping to secure network servers such as Proftpd.

8. Conclusion:

Our standalone smart seed generator enhances Nyx-Net's FTP server fuzzing with valid, protocol-aware seeds that overcome the limitations of generic inputs and Nyx-Net's regex-based specification. Based on Nautilus's grammar-based engine and an RFC 959-compliant grammar [4, 5], the system generates raw seeds, converted to Nyx-Net's .bin format, that ensure stateful compliance for servers like Proftpd. By requiring rules like PORT with values between 0-255 and RNFR and RNTO [5], our approach achieves superior seed validity (99.4% acceptance) and competitive coverage (19.0% edge, 25.7% line), addressing Nyx-Net's dependence on high-quality seeds as flat byte strings. But the Authors' Seed installation's detection of 215 crashes in 60 minutes, compared to zero for our grammar-based seeds, suggests that their seeds may have been curated to prioritize crash-inducing inputs, leveraging diversity and some invalid commands to uncover vulnerabilities.

While empirical validation by means of large-scale fuzzing campaigns is needed, the design lays out a pattern for scalable structured seed generation that can be applied to other protocols like SMTP or HTTP. The contrast between our grammar's high validity and the Authors' Seed's crash detection shows that validity has to be weighed against diversity of exploration. Future development of the grammar, such as for FTP extensions and varied inputs, could bolster crash detection without compromising validity, further optimizing the system's efficiency at exposing vulnerabilities. This work supports network security testing, establishing the foundation for robust protocol fuzzing and safer server implementations, with possibilities for ongoing optimization of seed generation methods based on insight from carefully curated seed sets like those potentially utilized by the authors.

References:

1. Van-Thuan Pham, Marcel Böhme, and Abhik Roychoudhury. AFLNET: A Greybox Fuzzer for Network Protocols. Monash University and National University of Singapore, 2019.
2. Sergej Schumilo, Cornelius Aschermann, Andrea Jemmett, Ali Abbasi, and Thorsten Holz. *Nyx-Net: Network Fuzzing with Incremental Snapshots*. In Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8 2022, Rennes, France. ACM, New York, NY, USA. https://doi.org/10.1145/3492321.3519591
3. Sergej Schumilo, Cornelius Aschermann, Ali Abbasi, Simon Wörner, and Thorsten Holz. *NYX: Greybox Hypervisor Fuzzing using Fast Snapshots and Affine Types*. Ruhr-Universität Bochum, 2021.
4. Cornelius Aschermann, Patrick Jauernig, Tommaso Frassetto, Ahmad-Reza Sadeghi, Thorsten Holz, and Daniel Teuchert. *NAUTILUS: Fishing for Deep Bugs with Grammars*. In NDSS Symposium 2019, 24–27 February 2019, San Diego, CA, USA.
5. Postel, J. & Reynolds, J. "File Transfer Protocol", RFC 959, October 1985.
6. RUB-SysSec. ProFuzzBench: A Benchmark for Stateful Protocol Fuzzing. GitHub Repository, Commit ab5078b3eda2f433eaac5fd3d6c640f7ffc85e72, 2022. Available at: https://github.com/RUB-SysSec/nyx-net-profuzzbench/tree/ab5078b3eda2f433eaac5fd3d6c640f7ffc85e72. Accessed: April 29, 2025.