Verify-Pro: A Framework for Server Authentication using Communication Protocol Dialects

Kailash Gogineni, Yongsheng Mei, Guru Venkataramani and Tian Lan
Department of Electrical and Computer Engineering, The George Washington University, Washington, DC, USA
{kailashg26, ysmei, guruv, tlan}@gwu.edu

***Abstract*—Customizing program binary and communication features is a commonly adopted strategy to counter network security threats like session hijacking, context confusion, and impersonation attacks. A potential attacker may have enough time to launch an attack targeting these vulnerabilities by rerouting the target request to a malicious server or hijacking the traffic. This paper presents a novel system *Verify-Pro*, a framework for server authentication using communication protocol dialects, to customize the communication features, enforce *continuous authentication*, detect the adversary, and prevent sensitive information leakage. Specifically, we leverage a machine learning approach (pre-trained neural network model) on both client and server machines to trigger a specific dialect that dynamically changes for each request (e.g., *get filename in FTP)*. Then, a decision tree algorithm is developed to automatically detect the adversary and terminate the entire session if the message is from an adversary. We implement a prototype of Verify-Pro and evaluate its practicality on standard communication protocol: FTP (File Transfer Protocol) and present a case study of the internet of things protocol MQTT (Message Queuing Telemetry Transport). Our experimental results show that by sending misleading information through message packets from an attacker at the application layer, the recipient can identify whether the sender is genuine or spoofed, with a negligible overhead of $< 1\%$.**

***Index Terms*—Program customization, Protocol dialects, Machine Learning, Network security, Authentication.**

I. Introduction

Communication protocols form the backbone of distributed computing infrastructure, where applications rely on data transfers to execute their tasks. It is, therefore, critical to preserve their security to avoid adversaries from exploiting any loopholes, bugs, and misconfigurations inherently embedded in the relevant software services. Numerous attacks in this threat space have been widely studied in the past- examples include obscuring network sources [1], [2], impersonating genuine sites [2], [3], MitM attacks through hijacking the request packets [4], where the attackers can easily launch them remotely without establishing a physical connection to their victims. In 2020, Barracuda researchers reported that conversation hijacking had increased 400% in 4 months [5]. Also, most legacy systems, including naval assets, are vulnerable to cyber attacks. To boost the security of such legacy systems, automated techniques that let the network protocols *adapt and transform* would be critical to achieve secure communication.

In many communication protocols (including the implementation of most popularly used protocols such as FTP [6], HTTP [7] & MQTT [7]), authentication typically occurs prior to the start of the session and this leaves them vulnerable to the communication protocol's attack surface. To counter them, we seek techniques that would ensure **continuous authentication** for every request in a session through cleverly leveraging **application layer features**.

Existing methods are limited to increasing complexity against potential attacks because low-level system properties (e.g., IP address [8], [9], TCP three-way handshakes [10], port numbers and proxies [9]) offer limited degree of freedom for mutation. Due to the above reasons, we design Verify-Pro with protocol dialects that leverage application layer properties and dynamically trigger a dialect to minimize the application attack surface. The problem is further complicated in insecure communication protocols like FTP, HTTP, and MQTT, as the data is transferred via plain text. The extensive use of these protocols and lack of continuous authentication for every request motivated the need for continuous authentication. For instance, the biggest file-sharing companies like *Box.com* and *BrickFTP* use FTP for their services because of its compatibility with legacy systems [11].

In this work, we present **Verify-Pro**, a framework that performs *Protocol Feature Customization (PFC)* by creating protocol dialects for each request in the session to improve the overall system security and maintain the core functionality of the underlying communication protocol. In this paper, we define ***protocol dialect*** *as variations of a standard protocol implementation at the binary level to incorporate additional security measures. Variations can be in the form of mutating message packets, generating different request-response transactions based on a few environmental conditions.*

Verify-Pro consists of three major modules: (1) Protocol dialects (PDs), (2) Dialect Decision Mechanism (DDM), and (3) Server Response Verification (SRV). The PDs module comprises several customized transactions used for communication between the client and server. When a command (e.g., *get file.txt* in FTP Protocol) is triggered by the client to retrieve a file from the server storage, the DDM module in the client is activated, and the request is fed as input to its neural network and a response dialect 'D_i' is determined for future verification. We note that the dialect selection must be unpredictable to eavesdroppers to prevent the hijacking attacks. To this end, we deploy a pre-trained neural network model on both client-server systems equipped with a customized design. In contrast to the shared key, the proposed mechanism induces the ability to dynamically and randomly change the indexing of dialect

for both client-server systems.

In addition, the DDM module provides a more secure way to trigger a dialect and use that dialect as the handshake to induce the system complexity (for the attacker) and resiliency by randomly changing the dialects. The strategies applied to the neural network are: 1. **Uniform distribution of dialects** offers an advantage in making it hard for the attacker to reverse engineer the neural network (guess the dialect number) as all the dialects are evenly distributed across the sample requests. 2. **Dialect selection based on cost** property offers a flexible neural network model to trigger the dialect with less cost and make the system more efficient (least cost for a dialect results in that dialect 'D_i' predicted more frequently across the sample requests). 3. **Consolidated loss** includes a trade-off factor 'a' which decides the sensitivity to the above-described properties. Since the client needs to verify the server's dialect in which the response was dispatched, the SRV module on the client side verifies if the server responds to the request using the 'correct' dialect 'D_i'. To our knowledge, this paper is the first to use a neural network as a decision mechanism (DDM) in triggering the dialects that dynamically change the transactions for each request.

The main contributions of our work are as follows:

- We propose Verify-Pro, an automated framework for applying communication protocol dialects as fingerprints to authenticate servers. We harness different protocol dialect implementations and leverage them to create unique responses that help authenticate servers during communication and improve security.
- Verify-Pro uses a neural network model to select a unique dialect for response to be used for each request. The motivation behind the neural network model is to deploy a customized mechanism for the selection of dialect and avoid reverse engineering attacks by adversaries.
- We design and implement Verify-Pro prototype on FTP & MQTT, and evaluate its effectiveness using dialects as fingerprints on a real-world setup. Our evaluation results show that Verify-Pro can successfully counter the attack surface and improve the security in File-sharing system.

II. Threat Model

From an offensive perspective, the attacker's objective is to send an unwanted or malformed response to a target machine. We consider a threat model in which an adversary can actively divert the requests and responses exchanged between the client and server machines. For example, the active adversary can replay, use proxies, intercept, fabricate new messages and stop messages from reaching their destination (by sending the request to a malicious server)-request hijacking. In particular an attacker can launch the context confusion attacks by setting their base station in the same LAN as the victims, being able to reroute the encrypted traffic (request), where the MitM attacks rely on shared TLS certificates [4], [12]. Bugs related to malicious response or lack of continuous authentication are reported in CVE-2019-9760 and CVE-2021-41638 on FTP.

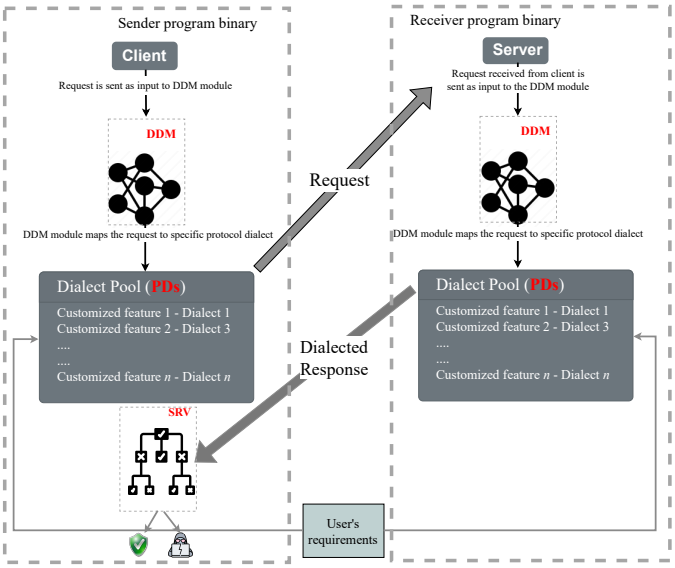


Figure 1: Verify-Pro System Diagram.

We make the following assumptions to support the Verify-Pro system: 1) The responses sent from server to client can be malformed or replayed, and the request sent by the genuine client can be hijacked to a flawed server. 2) We further assume that the attacker has no means to access or directly compromise the software, storage, and data structures of the neural network executing on the client and server.

III. System Design

Verify-Pro consists of three major modules: (1) Protocol dialects (PDs), (2) Dialect Decision Mechanism (DDM), and (3) Server Response Verification (SRV). In Figure 1, we provide an illustration of the Verify-Pro system diagram.

1) Phase 1: Protocol dialects (PDs): At the core of creating customized protocol dialects, the important problem is to understand the variations in the handshakes. We perform PFC by implementing customized transaction functions (e.g., mutating the message format variations) in the communication protocol. We leverage these handshakes, cross-graft them into an existing communication protocol and use them as fingerprints to verify the identity of the response sender. We design fifteen dialects (in FTP) as a proof-of-concept, and they are deployed in both client and server machines to communicate effectively in one of the dialects triggered for each request. We present few handshakes of FTP protocol in Figure 2 & 3a (*due to page restriction we only provided eight dialects*). The server's response to different dialects is highlighted in green. Each dialect has a unique message structure that helps the client identify the dialect number used by the server to send its response. The protocol dialects are spawned as different versions of a protocol deployed into the single communication protocol binary. Furthermore, deploying dialects as threads gives us an added advantage of minimal overhead and less cost, as the triggering of each dialect happens in milliseconds.

2) Phase 2: Dialect Decision Mechanism (DDM): We implement the DDM module as a deep neural network which has input as the '$request$' (e.g., *get file.txt* in FTP protocol), label as the dialect number (ranges from 1 to 15). The output of the DDM module will be used as the dialect number to start the communication, and the customized handshake is initiated. We make use of the NLP corpus of words https://norvig.com/ngrams/ for creating a customized dataset. Our neural network requires the input to be the '$request$' of the communication protocol. The dataset only includes the list of requests (e.g., *get filename*) as the model is constructed in an unsupervised setting, as it contains a large set of 150K unlabeled sample requests. We note that the dialect selection must be unpredictable to eavesdroppers in order to prevent the MitM- session hijacking and context confusion attacks [4]. To this end, we deploy a pre-trained neural network model on both client-server systems equipped with a customized design with the following strategies:

Uniform distribution of dialects offer an advantage in making it hard for the attacker to reverse engineer the neural network (guess the dialect number) as all the dialects are evenly distributed across the sample requests. We used entropy maximization in the loss function for training. Here, the $P(y_i)$ represents the occurrence of particular dialect number of request y_i (computed as the number of occurrences of requests with a particular probability i divided by the number of all requests of that particular family), log2 is a logarithm with base 2, and M is the total number of dialects (classes). This property makes the neural network model resilient as the attacker will have to invest time and effort to predict or inverse the model as all the dialects have an equal probability of occurring.

$$Uniformity\ \ loss\ (l1):\ min\sum_{i=1}^{M}(P(y_i)log_2(P(y_i))) \quad (1)$$

Dialect selection based on cost property offers a flexible neural network model to trigger the dialect with less cost and make the system more efficient (least cost for a dialect results in that dialect number D_i predicted more frequently across the sample requests). We assume cost C_i (where C_i is the cost of each dialect) for each dialect and $P(y_i)$ is the probability distribution of choosing a dialect, then we aim to minimize the sum of $P(y_i) \times C_i$ which is the expected cost. M is defined as the total number of dialects (classes). This property offers the flexibility to make custom predictions based on the cost individually assigned to each dialect. For example, to confuse the middle attackers, we intended to communicate in dialect 4 with the highest chance of prediction, whereas dialect 8 should have the least chance. In this case, we assign a higher value as the cost to dialect 4, whereas the least number (cost) for dialect 8. After training, the model would predict dialect 4 with a high frequency instead of uniform distribution of dialects. Customization with cost makes the system more flexible to revise the prediction frequently and confuse the attackers.

$$Cost\ \ based\ \ dialect\ \ loss\ (l2):\ min\sum_{i=1}^{M}(P(y_i)(C_i)) \quad (2)$$

Consolidated loss: We use the formula from equation (3) to calculate the consolidated loss using a trade-off factor 'a', in range [0,1]. The combination of these losses l1 (1) & l2 (2) proved to be effective such as, by varying the 'a' value, the prediction of dialects will gradually change from *'finding the dialect with low cost'* to *'evenly distributing the dialects'* across the sample requests and allows a customized design for prediction of dialects.

$$Consolidated\ \ loss\ (l3):(a \times l2) + ((1-a) \times l1) \quad (3)$$

3) Phase 3: Server Response Verification (SRV): After receiving the server's response, the client sends the response as the input to the decision tree, which verifies the response structure (format in which the packets are sent) of the sender's response (i.e., server's response) to avoid overlapping with any dialect's response or any malicious response. In our decision tree model, an input x is traversed in the tree learned on Δ. For example in FTP protocol, we show our server response - the input is:

Input: 'P1: command/ P2: filename, length of filename'

For the learning purpose, we convert the input into sized vectors, and we consider the data types with fields separated by ',' and the packets separated by '/'. Packet 1 (P1) has a string as the first field for the above input, and packet 2 (P2) has a string as the first field and an integer as the second field. Since we verify the data type of each field in the packet and the structure of the packet, this compels us to create a data set of 150K samples with random strings and integers according to each dialect response pattern. Following the vector conversion, the input is traversed through the pre-trained model, and a class (dialect number) is predicted with which that response structure matches. After verifying the message pattern received and confirming the dialect, we also verify the values of fields, as the client already knows some information about the message it will receive (such as command, length of command). We used CART [13] decision tree for making the decisions to detect the adversary and terminate the communication channel.

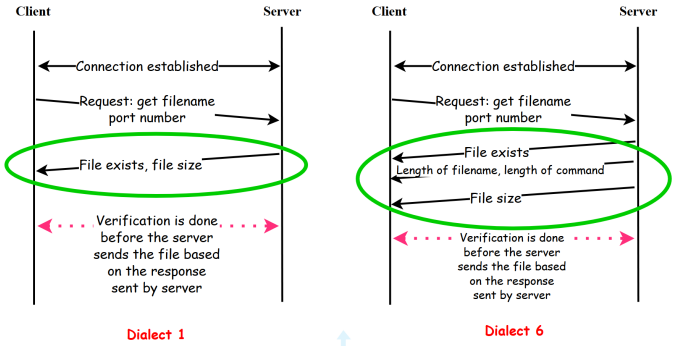


Figure 2: Request-Response of Dialect 1 and Dialect 6 in FTP.

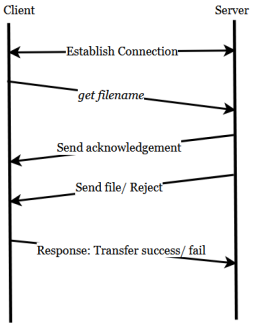
IV. Evaluation

We prototyped Verify-Pro on File Transfer Protocol (FTP) program binary as a proof of concept. We customize the

FTP protocol on client-server systems to include 15 customized transactions-protocol dialects to provide continuous authentication for each request in the session. In Figure 3a, we provide the list of all the protocol dialects. We create a variety of dialects by changing the communication rules such as packet mutation, generating different request-responses, communicating in binary format. For example, in dialect 5, the communication happens with numbers - 1 (means file exist) & 0 (means file does not exist), Dialect 7- divides a single packet and sends the information in sub-packets. Dialect 4 sends the file size in two separate packets, and in the same way, all the dialects have a unique response structure before the file is transferred. Our main aim is to detect the adversary in the initial phase of a handshake so that the client can terminate the connection without even entering the file transferring process.

Dialect number	Request-Response pairs
1	Request: get filename Request: Port number Response: File exists, file size File sharing - Only after client verifies server's response
2	Request: get filename Request: Port number Response: file size, file size ▸ file size sent two times Response: Connection Closed
3	Request: get filename Request: Port number Response: File exists, file size, file name
4	Request: get filename Request: Port number Response: File size/2 Response: Remaining file size
5	Request: get filename Request: Port number Response: 1 (file exists), length of filename, length of command Response: File size
6	Request: get filename Request: Port number Response: File exists Response: length of filename, length of command Response: File size
7	Request: get filename Request: Port number Response: File exists Response: Length of filename Response: Length of command
8	Request: get filename Request: Port number Response: File exists, size of file, filename,command

(a) File Transfer Protocol Dialects



(b) Timing diagram of GET

Figure 3: FTP dialects and timing diagram

Experiment Setup: Our experiments are conducted on a 3.20 GHz Intel(R) Core(TM) CPU i7-4790S machine with 15.5 Gigabytes of main memory. The operating system is Ubuntu 18.04 LTS. We choose FTP & MQTT as our preliminary benchmarks. We implement the prototype of Verify-Pro based on communication protocols to utilize the infrastructure of dialect library, dialect pattern and decision tree based authenticity verification in the form of several python modules. We used the library scapy [14] and wireshark [15] for the communication protocols implementation primitives and packet capturing. Our code consists of 4800 lines of python code compared with the default python implementation of target program binaries (FTP & MQTT), together with 300 lines of python code for automation and testing the communication protocols. To demonstrate the precursory experiments of Verify-Pro against the attack surface mentioned above, we created a setup containing proof-of-concept implementation with the Verify-Pro knowledge base and according to the system (experimental) resources available.

DDM module training process: Our model is a simple deep neural network, which is able to map the input feature vectors $x = x_1, ...x_n$ *(converting the 'request' into high-dimensional vectors)* consisting of n samples to an output y_i *(which is the dialect number for a given request)*. The input of the neural network has a size of $n = 100$ *(vector for each request)*, fed as a high dimensional feature vector. The model has two hidden layers with 128 neurons each and 'relu' activation function in each layer but the last layer has n neurons *(n represents the number of dialects)* with 'softmax' activation function. The ADAM optimizer was used for the training process. The models were implemented by using Python3.6 and Keras [16] with Tensorflow backend [16]. We used 15 neurons in the last layer, 0.0001-learning rate, 100 epochs for cost loss and 100 epochs for entropy loss, 128-batch size, trained and tested the model with 80%-20% ratio as the system configuration.

SRV module training process: We use the sklearn [16] package in Python, to design a decision tree (CART [13]) without manually specifying the rules for decision making. To train a CART decision tree classifier, given a training dataset, the decision tree is obtained by splitting the set into subsets from the root node to the children node. The splitting is based on the rules derived by the Gini index. In our scheme, we only consider the pre-trained decision tree model on the client-side to verify the authenticity of the server's response. We train the classifier with *max depth = 7*, trained and tested the model with 80%-20% ratio as the system configuration.

A. Customizing FTP

FTP [6], [11] is a standard communication protocol used for data transfer between client-server systems. As a target protocol for our proof-of-concept evaluation, FTP has two main benefits: (a) a light-weight network protocol having finer performance, flexibility, and ease in testing, and (b) It has less complexity in design, supports in customizing the protocol at the binary level for providing additional security measures. FTP packet format contains IP header, TCP header and FTP message (file). When a request '$get \quad filename$' is sent to the server, the default FTP protocol has a request-response handshake (shown in Figure 3b). After applying Verify-Pro on FTP, the PDs module comprises a dialect library on client-server machines. DDM module on both client-server systems is used to choose a dialect 'D_i' for each unique request 'R_i'. Request 'R_i' (undialected request) is sent to the server, and the client awaits the response from the server to verify the server identity. On the server-side, utilizing the request 'R_i' received from the client, the server uses the DDM module to determine the dialect number 'D_i' to send a dialected response '$resp$' to the client. In turn, the client uses the SRV module to validate the server's response.

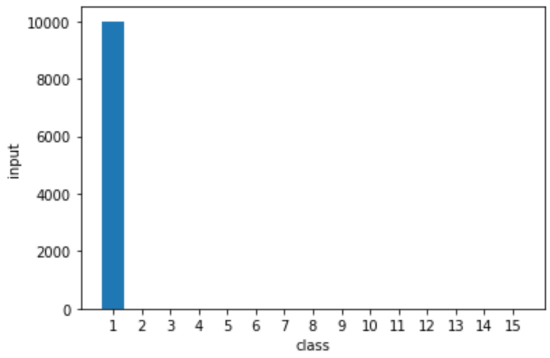
1) Analysis of Table Ia: In the end, we evaluate the execution overhead of Verify-Pro, by transferring a file of 20 bytes with dialect eight and compare the results with standard FTP (deployed dialect-8 template). Execution overhead metrics:

System time*: Time recorded from the user login to a 20 byte file transfer in seconds for Dialect 8 (Verify-Pro) and FTP (deployed dialect-8 template).*

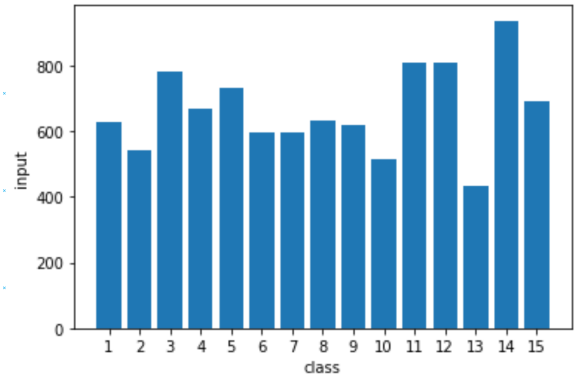
DDM time*: Time logged from triggering of the user request to the prediction of dialect number.*

SRV time*: Time logged from feeding the response as input to the Decision tree until outputting the dialect as confirmation.*

To be concrete with our evaluation, we also check the overhead of the modules which are added when compared with the original FTP implementation. We present the overhead of DDM and SRV modules. Since both these modules have pre-trained models with a size of 12MB for neural network model and 7KB for decision tree models, the execution time of these modules is negligible. Besides the overhead of DDM, SRV modules, the remaining overhead incurs when the client verifies information of each field such as command, filename, etc. Our PDs module does not incur any overhead. The dialects are created as threads such that only one instance 'C_i' will be activated for a given dialect 'D_i' and for the unique request 'R_i'. To avoid potential statistical bias, we execute the experiment multiple times and compute the average overhead (see Table Ia). Precisely, we conclude that the addition of PDs, DDM, SRV modules incurs 0.536% overhead (from system time), which is trivial; in turn, the addition of these modules enforces continuous authentication. Furthermore, the run-time overhead for all the protocol dialects is $< 1\%$ (on average), which is negligible.



(a) *a = 1 for l2 loss & a = 0 for l1 loss*



(b) *a = 0 for l2 loss & a = 1 for l1 loss*

Figure 4: Variations of charts with the trade-off factor a. This graph shows dialect numbers on $x - axis$ & requests on $y - axis$.

2) Case Study: FTP: To demonstrate the effectiveness of our Verify-Pro tool, we create an attacker FTP server that implements a spoofed (or impersonate) dialect of the FTP. Once client and server (equipped with Verify-pro countermeasure) systems are on the communication loop, the target request is sent from a genuine client to a malicious server (connection reset) by an MitM attacker [17] to launch the context confusion attacks [4] and origin issues [18]. The malicious server can start fabricating and sending malicious responses to the genuine client. The communication channel between genuine client and server systems is shown in Figure 5. The client sends the ***get joyal.txt*** command (FTP passive implementation-client sends the port number) to retrieve the file from server storage. Communication happens with dialect-4 as the file size is divided and sent in two packets. Figure 6 shows the communication between client and attacker server machines to retrieve ***loka.txt*** file from server storage. It is obvious that the attacker server (as PoC, we used a shadow neural network with 64 as batch size to show the dialect mismatch, whereas our genuine server uses 128 as batch size) finds it difficult to understand the dialect evolution pattern generated by the DDM module. The client-server systems share a different neural network, subsequently selecting different dialects for the same request and the handshake is aborted with attacker server sending a response ***not found*** & no file is transferred. The malicious server fails in the dialect evolution phase, as continuous authentication is performed for every request in the session and the client protocol dialect pattern changes dynamically for every request. From the preliminary experiments, we believe that our method helps to safeguard the communication protocols by countering the attack vectors such as rerouting the target request [4], malicious messages. In particular, using the customized neural network helps us choose the dialects and randomly change the pattern with significantly less cost (training the model: 20 seconds).

3) Analysis of Table Ib: We present the attack success probability (i.e., 1/(no.of dialects)) to predict the correct dialect. The increase in the number of dialects results in the attacker having significantly less probability to predict the correct

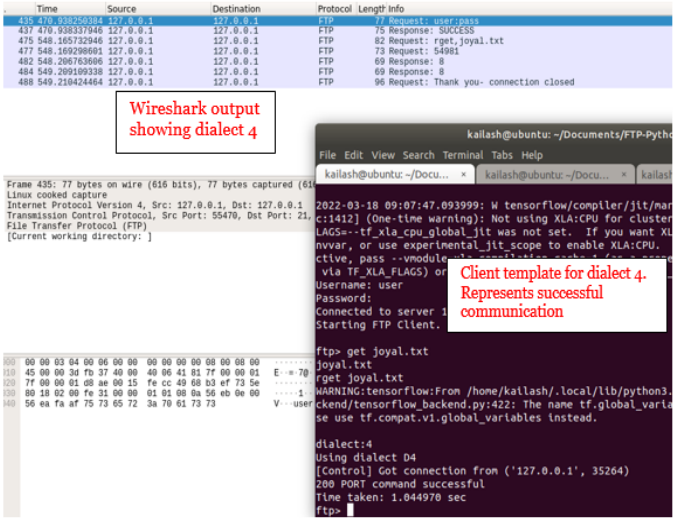


Figure 5: Scenario-1: Client & Server communication channel

Performance analysis of Verify-Pro (FTP) vs. FTP		
Performance Index	FTP	Verify-Pro (FTP)
CPU% Utilization	<1%	1%
System time/sec	43.871 sec	44.106 sec
DDM model time/sec	N/A (No DDM)	0.0723 sec
SRV model time/sec	N/A (No SRV)	0.000525 sec (only on client)

(a) Performance Metrics.

Probability for the attacker in predicting the correct dialect		
Number of Dialects	Property	Success probability (%)
8	Uniform Distribution (l1)	12.5%
50	Uniform Distribution (l1)	2%
75	Uniform Distribution (l1)	1.33%
100	Uniform Distribution (l1)	1%
1000	Uniform Distribution (l1)	0.1%

(b) Attacker Success Probability.

MPD vs. Verify-Pro (FTP)	
Prototype	Execution overhead (%)
MPD [19]	4.43%
Verify-Pro (our work)	1%

(c) Execution Overhead (%)

Table I: Evaluating the performance, attacker success probability & overhead of the designed Verify-Pro in FTP.

dialect. For instance, the attacker's success probability with the uniform distribution loss (l1) and the number of dialects (=100) is 1% and will be even smaller when the dialects are increased. On the other hand, when the cost-based loss (l2) is used, only a specific dialect will be predicted more often and obfuscates the attack surface. We observed zero correlation between the two sample datasets by experimenting with the two different testing datasets with 10K sample requests, each using the uniform distribution and dialect selection with cost properties. As our work is a PoC, we mainly show the analysis of FTP protocol with fifteen dialects. Our customization framework provides the broad capabilities that can be incorporated into any communication protocol and provides an automated program feature selection using the learning-based approach (DDM). Further increasing the protocol dialects via deep learning or formal analysis provides interesting directions for future work.

4) Trade-offs of DDM module: In this section, we use a trade-off factor a to show the flexibility of the DDM module to adapt to different user requirements. To minimize the compromise of communication between ships and naval bases, we consider two base stations (*bs-1 & bs-2*) to use a customized transaction (protocol dialect-1) to start the communication. In that case, *bs-1* sends the request to *bs-2* and then *bs-2* (if it's genuine) feeds the request 'R_i' to the DDM module and responds in a dialect 'D_i'. After receiving the message, *bs-1* deconstructs the message from *bs-2* and verifies the dialect number 'D_i' of the sender's response. For instance, in scenario-1 (Fig 4b) to obfuscate the eavesdropper, our prototype can be effective to minimize the cyber attacks by communicating in different dialects by assigning the full priority to the *l1 loss*. This practice eases the message exchange between the two parties by using diverse set of protocol dialects. On the other hand, in scenario-2 (Fig 4a), the base stations *bs-1 & bs-2* can communicate in a single protocol dialect pattern for the entire session by assigning full priority to the *l2 loss* and specifying highest cost to the dialect according to user requirements. The interesting find from Figure 4b is that, as the a value decreases from 1 to 0, we observed that all the dialects are *evenly distributed* in such a way that every dialect has an equal likelihood of happening.

Figure 6: Scenario-2: Client & Attacker-Server channel

B. *Case study: Applying Verify-Pro knowledge base on MQTT*

MQTT [7] is a standard lightweight IoT messaging protocol that is used for IoT devices. It is a binary-based protocol, which has ***command*** and ***command acknowledgment*** format. The MQTT protocol payload carries the data such as binary, ASCII data, etc. It uses packets of small size, hence offers benefits for low bandwidth applications. MQTT publish packet (Figure 7.) contains a fixed header (including control header), variable header, and payload on the application layer [20]. The client will send a $publish$ packet to the server, and the server will respond with '$pub-ack$' message. The timing diagram of connect packet and publish packet is shown in Figure 8. MQTT connect packet (Figure 9.) contains the 2-byte fixed header (always present), variable header and the payload on the application layer. We programmed a standard MQTT client and broker (server) and applied our Verify-Pro knowledge base on them. We customized the *publish and connect* packets in the MQTT protocol and we assume the client has to prove its authenticity to the server.

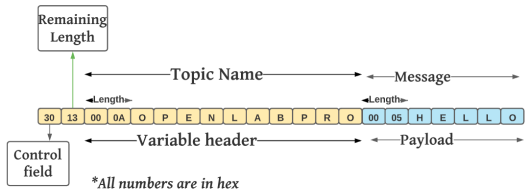


Figure 7: MQTT publish packet format [20].

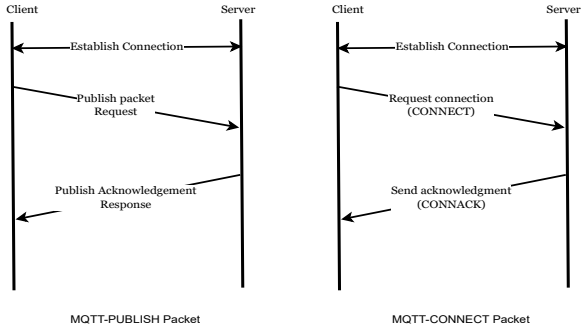


Figure 8: Timing diagram of MQTT *publish* & *connect* packets.

Table II presents the protocol dialects applied on MQTT protocol. Feature customization like including external features (command name, length of command, etc.) can be cross-grafted to the MQTT handshake to generate diverse set of dialects. As can be seen from Table III, continuous authentication can be implemented on MQTT with minimal additional overhead ($< 0.8\%$), less cost and can be deployed incrementally, hence making it a scalable solution. Given its security benefits, we believe that Verify-Pro can function as an additional strong protection layer in conjunction with existing authentication mechanisms. Execution overhead metrics:
System time*: We compute the average time for all the dialects (MQTT). We only recorded the system time for one complete handshake. More precisely, the time the dialect is triggered and gets the confirmation of dialect from another machine.*

Dialect name	Mutations done
1) Header shuffle (publish packet)	Topic and message fields are switched
2) Mutation of payload (publish packet)	A default MQTT has 1 topic and 1 value in each publish packet, but this variant publishes the message with 2 topics and 2 values to save bandwidth
3) Mutation of payload (publish packet)	Remove the topic field and send both topic and value as a single packet
4) Drop bytes (connect packet)	Drop keep-alive bytes
5) Field switching (connect packet)	Protocol version and the connect flags are swapped

Table II: MQTT protocol dialects.

Performance Index	MQTT	Verify-Pro (MQTT)
CPU Utilization	<1%	<1%
System time/sec	4.293	4.325
DDM model time/sec	N/A (No DDM)	0.0715 sec
SRV model time/sec	N/A (No SRV)	0.000617 sec (only on server)

Table III: Performance analysis of MQTT protocol

C. Impact on program security

An adverse side-effect of communication protocols, especially for FTP and MQTT, is that authentication typically occurs before the start of the session. Even without removing any required functionality, the software can also be transformed to be more efficient. Program Feature Customization (PFC) is one such technique for such situations to effectively protect the system by enforcing continuous authentication to avoid the adversary compromising a server which could lead to disastrous outcomes. Conversely, legacy systems supported communication protocols contain multiple user-desired features that may be rarely used (*unnecessary features from functionality standpoint*), which results in software bloat. Such rarely used protocol features could be exploited by malicious parties as back-door entries to gain access to sensitive information.

For instance, performing feature elimination (remove the fields) on the *keep-alive* bytes (Table II) in MQTT *connect* removes the CVE-2020-13849 vulnerability by curtailing the

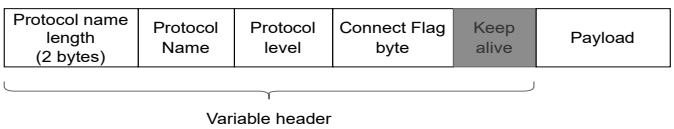


Figure 9: MQTT connect packet format [20].

Denial of Service (DoS) attacks. We survey the known CVEs of different programs that can be uprooted by our Verify-Pro knowledge base. For instance, in MQTT, i) the CVE-2020-6881, known as DoS vulnerability can be used to detect abnormal messages by crafting the messages according to a dialect pattern in Table II and enforcing continuous authentication; ii) the CVE-2021-21967 which can cause denial of service, can be negated by imposing continuous authentication to obfuscate the middle attackers and changing the communication rules for the *publish* packet; iii) CVE-2020-27220, CVE-2021-3618, which have the weak authentication bug, can terminate the connection with the attacker even after advancing the *username and password* stage by enforcing continuous authentication. The CVE-2021-41637, CVE-2021-41638, CVE-2021-41638 (weak authentication problem), CVE-2016-4971, CVE-2019-9760 (traffic hijacking bug: redirect target request to attacker server) in FTP can be nullified by enforcing continuous authentication for every request in the session. In total, we found 10 CVEs in MQTT and FTP network protocols during the 2019-2022 and one from 2016. Not all the vulnerabilities can be directly eliminated by our PFC as some vulnerabilities are the applications and program binaries of the standard network protocols. As a preliminary experiment, we performed feature cross-grafting, feature elimination, and protocol dialecting and used a custom-built learning-based approach to the network protocol implementation. In all the cases, Verify-Pro is validated, tested, and verified to ensure that it does not break the existing protocol functionality. Further, to protect network communication's privacy and data integrity, TLS and SSL can be wrapped with the protocol dialects to boost the security of the standard protocol.

V. Related work

In this section, we first discuss the existing efforts on protocol customization and program binary analysis to reduce vulnerabilities in real-world programs to develop the secured version of the protocol binary. Further, we briefly discuss how Verify-Pro is related to previous work in this research area.

Protocol Customization. Existing approaches focus on communication protocol mutations by leveraging lower-level system configurations (e.g., IP address) [9], [10], [21]–[23]. Different from these works, our scheme mainly focuses on leveraging the application layer features in designing protocol dialects and enforces continuous authentication for every request in the session. Ghost-MTD [24] proposed a protocol mutation scheme that uses a previously shared one-time bit sequence (OTBS). In this mechanism, the protocol variation pattern should be predefined between the user and service module of the server system. The closest work to this paper is MPD [19], which dynamically customizes a communication protocol into various protocol dialects by leveraging the application layer properties to create a moving target defense with an execution overhead of 4.43% (includes randomization, pseudo-random function, and consistent hash mapping, which suffer overhead problems). Our approach, however, uses a DDM module to trigger a dialect instead of using randomiza-

tion properties by achieving 77% less computation overhead than MPD [19]. In contrast to the previous works, some works focus on fingerprinting methods [8], [25], [26]. For instance, Hfinger [26] a malware fingerprinting tool that extracts the information from the parts of the request such as URI, protocol information, headers, etc., and generates fingerprints.

Program binary analysis. Binary analysis and program rewriting techniques have been widely employed in the security research to remove undesired code in the program binary to reduce the attack surfaces [27]–[30]. In MORPH [28] and Hecate [29], deep learning-based methods are used for trace analysis, which provides insights into our design by deploying a deep learning-based decision model to trigger a communication protocol variant and their constituent functions by mapping the requests to a program feature.

VI. Conclusion

We presented a novel framework Verify-Pro, which aims to use the dialects as fingerprints during communication and dynamically select them to enforce continuous authentication. Empirical results indicate that Verify-Pro can minimize the attack surface, assist in effective communication, and improve overall system security while incurring negligible execution overhead of $<1\%$ for the network protocols (FTP & MQTT) and achieves 77% less overhead compared to MPD [19].

Acknowledgements

This research is based on work supported by the US Office of Naval Research (ONR) under grant N00014-17-1-2786. Any opinions, findings, conclusions expressed in this article are those of authors, and do not necessarily reflect those of ONR.

References

[1] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[2] D. Rupprecht, K. Kohls, T. Holz, and C. Pöpper, "Imp4gt: Impersonation attacks in 4g networks," in *Symposium on Network and Distributed System Security (NDSS). ISOC*, 2020.

[3] R. Roberts, Y. Goldschlag, R. Walter, T. Chung, A. Mislove, and D. Levin, "You are who you appear to be: a longitudinal study of domain impersonation in tls certificates," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2489–2504.

[4] M. Zhang, X. Zheng, K. Shen, Z. Kong, C. Lu, Y. Wang, H. Duan, S. Hao, B. Liu, and M. Yang, "Talking with familiar strangers: An empirical study on https context confusion attacks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1939–1952. [Online]. Available: https://doi.org/10.1145/3372297.3417252

[5] D. MacLennan. (2020) Threat spotlight: Conversation hijacking. [Online]. Available: https://blog.barracuda.com/2020/01/16/threat-spotlight-conversation-hijacking/

[6] J. Postel and J. Reynolds, "File transfer protocol," 1985.

[7] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.

[8] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser, "Detecting spammers with snare: Spatio-temporal network-level automatic reputation engine." in *USENIX security symposium*, vol. 9, 2009.

[9] J.-H. Cho, D. P. Sharma, H. Alavizadeh, S. Yoon, N. Ben-Asher, T. J. Moore, D. S. Kim, H. Lim, and F. F. Nelson, "Toward proactive, adaptive defense: A survey on moving target defense," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[10] E. Al-Shaer, "Toward network configuration randomization for moving target defense," in *Moving Target Defense*. Springer, 2011, pp. 153–159.

[11] D. Springall, Z. Durumeric, and J. A. Halderman, "Ftp: The forgotten cloud," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 503–513.

[12] E. S. Alashwali and K. Rasmussen, "What's in a downgrade? a taxonomy of downgrade attacks in the tls protocol and application protocols using tls," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2018, pp. 468–487.

[13] L. Breiman, J. Friedman, C. Stone, and R. Olshen, "Classification and regression trees chapman & hall," *New York*, 1984.

[14] (2016) Scapy: the python-based interactive packet manipulation program & library. supports python 2 & python 3. [Online]. Available: https://github.com/secdev/scapy.git

[15] (1998) Wireshark: open-source packet analyzer. [Online]. Available: https://www.wireshark.org/

[16] M. J. Douglass, "Book review: hands-on machine learning with scikit-learn, keras, and tensorflow, by aurélien géron," 2020.

[17] Thiccmalware. (2018) Framework for man-in-the-middle attacks. [Online]. Available: https://github.com/byt3bl33d3r/MITMf.git

[18] M. Brinkmann, C. Dresen, R. Merget, D. Poddebniak, J. Müller, J. Somorovsky, J. Schwenk, and S. Schinzel, "{ALPACA}: Application layer protocol confusion-analyzing and mitigating cracks in {TLS} authentication," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 4293–4310.

[19] Y. Mei, K. Gogineni, T. Lan, and G. Venkataramani, "Mpd: Moving target defense through communication protocol dialects," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2021, pp. 100–119.

[20] (2019) Mqtt. [Online]. Available: https://openlabpro.com/guide/mqtt-packet-format/

[21] T. Ouyang, S. Ray, M. Allman, and M. Rabinovich, "A large-scale empirical analysis of email spam detection through network characteristics in a stand-alone enterprise," *Computer Networks*, vol. 59, pp. 101–121, 2014.

[22] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 127–132.

[23] K.-W. Kang and K.-W. Park, "Toward software-defined moving target defense for secure service deployment enhanced with a user-defined orchestration," in *Proceedings of the 2020 ACM International Conference on Intelligent Computing and its Emerging Applications*, 2020, pp. 1–5.

[24] J.-G. Park, Y. Lee, K.-W. Kang, S.-H. Lee, and K.-W. Park, "Ghost-mtd: moving target defense via protocol mutation for mission-critical cloud systems," *Energies*, vol. 13, no. 8, p. 1883, 2020.

[25] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A survey of https traffic and services identification approaches," *arXiv preprint arXiv:2008.08339*, 2020.

[26] P. Białczak and W. Mazurczyk, "Hfinger: Malware http request fingerprinting," *Entropy*, vol. 23, no. 5, p. 507, 2021.

[27] Y. Chen, S. Sun, T. Lan, and G. Venkataramani, "Toss: Tailoring online server systems through binary feature customization," in *Proceedings of the 2018 Workshop on Forming an Ecosystem Around Software Transformation*, 2018, pp. 1–7.

[28] H. Xue, Y. Chen, G. Venkataramani, T. Lan, G. Jin, and J. Li, "Morph: Enhancing system security through interactive customization of application and communication protocol features," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2315–2317.

[29] H. Xue, Y. Chen, G. Venkataramani, and T. Lan, "Hecate: Automated customization of program and communication features to reduce attack surfaces," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2019, pp. 305–319.

[30] H. Xue, Y. Mei, K. Gogineni, G. Venkataramani, and T. Lan, "Twinfinder: Integrated reasoning engine for pointer-related code clone detection," in *2020 IEEE 14th International Workshop on Software Clones (IWSC)*. IEEE, 2020, pp. 1–7.