symmetry

MDPI

Article

Program Behavior Dynamic Trust Measurement and Evaluation Based on Data Analysis

Shuai Wang *, Aiqun Hu, Tao Li and Shaofan Lin

School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China; aqhu@seu.edu.cn (A.H.); lit@seu.edu.cn (T.L.); linshaofan@seu.edu.cn (S.L.)
* Correspondence: 230209303@seu.edu.cn

Abstract: Industrial control terminals play an important role in industrial control scenarios. Due to the special nature of industrial control networks, industrial control terminal systems are vulnerable to malicious attacks, which can greatly threaten the stability and security of industrial production environments. Traditional security protection methods for industrial control terminals have coarse detection granularity, and are unable to effectively detect and prevent attacks, lacking real-time responsiveness to attack events. Therefore, this paper proposes a real-time dynamic credibility evaluation mechanism based on program behavior, which integrates the matching and symmetry ideas of credibility evaluation. By conducting a real-time dynamic credibility evaluation of function call sequences and system call sequences during program execution, the credibility of industrial control terminal application program behavior can be judged. To solve the problem that the system calls generated during program execution are unstable and difficult to measure, this paper proposes a partition-based dynamic credibility evaluation method, dividing program behavior during runtime into function call behavior and system call behavior within function intervals. For function call behavior, a sliding window-based function call sequence benchmark library construction method is proposed, which matches and evaluates real-time measurement results based on the benchmark library, thereby achieving symmetry between the benchmark library and the measured data. For system call behavior, a maximum entropy system call model is constructed, which is used to evaluate the credibility of system call sequences. Experiment results demonstrate that our method performs better in both detection success rate and detection speed compared to the existing methods.

Keywords: dynamic credibility; dynamic metrics; program behavior analysis; program structure graph analysis; behavior monitoring

Citation: Wang, S.; Hu, A.; Li, T.; Lin, S. Program Behavior Dynamic Trust Measurement and Evaluation Based on Data Analysis. *Symmetry* **2024**, *16*, 249. https://doi.org/10.3390/sym16020249

Academic Editor: Michel Planat

Received: 16 January 2024
Revised: 8 February 2024
Accepted: 15 February 2024
Published: 17 February 2024

1. Introduction

With the continuous improvement of industrial automation, industrial control systems have become an indispensable key infrastructure in various industries. Compared with traditional IT systems, industrial control systems are more concerned with real-time performance, reliability, and security. In industrial control systems, industrial control terminals are important components that typically involve devices such as sensors, actuators, and scheduling instruments, which can directly control production lines or other industrial processes. While the widespread application of industrial control systems has brought many security issues, recent security threat reports [1] show a significant increase in software vulnerabilities over the past decade, with most information security vulnerabilities coming from endpoint devices such as mobile devices and IoT smart devices that are vulnerable to attacks. These attacks can not only lead to production line paralysis but also disrupt normal industrial production, causing economic and human losses [2]. Therefore, the security threats to industrial control terminals have received high attention.

The common attacks on industrial control terminals are code reuse attacks and mimicry attacks. Code reuse attacks achieve the attack goal by directing the program control flow

to special code snippets, which have strong destructive power. To detect code reuse attacks, techniques such as stack randomization, code signing, and address space layout randomization can be used, but these techniques produce higher latency, so they are not highly applicable in industrial control scenarios.

Mimicry attacks [3] refer to attackers using legitimate execution call sequences to achieve attack goals by modifying parameters within the called function. Techniques such as parameter discovery [4] or data flow analysis [5] can be used to detect these attacks. However, it is difficult to ensure full coverage of all input parameters in the application program, and the parameter discovery process is tedious and time-consuming. Data flow analysis detection produces a large number of false positives when dealing with exceptional situations and also incurs high system load and operating costs. Therefore, both technologies have lower accuracy in detecting mimicry attacks.

Based on the problems with current attack detection techniques for industrial control terminals, there is a requirement to improve the running safety of industrial control terminal applications from both the accuracy and real-time aspects. Static analysis [6] is based on the source code of the program itself and can obtain accurate information. Dynamic measurement [7] can monitor and statistically analyze specific metric objects during program execution, avoiding the resource consumption and time cost associated with analyzing the entire program. Therefore, this paper combines static analysis and dynamic measurement, which divides program execution behavior into function call behavior and system call behavior, obtains a complete and lightweight function call sequence through static analysis, and establishes a benchmark library. We design a sliding window execution sequence measurement method and a maximum entropy system call model based on dynamic measurement to measure program behavior, which matches and evaluates real-time measurement results based on a benchmark library. Our proposed mechanism can achieve symmetry between the benchmark library and the measured data, and improve the accuracy and real-time performance of dynamic credibility evaluation.

In summary, this paper conducts research based on program execution flow to address the issues of code reuse attacks and mimicry attacks on industrial control terminals. We design a real-time dynamic credibility measurement evaluation mechanism that combines static analysis and dynamic measurement techniques. This mechanism is a novel and effective industrial control terminal security detection technology with high detection accuracy and real-time performance. Its application will help to ensure the security of industrial control terminals and further improve the reliability and stability of industrial automation systems.

The main contributions of this paper are as follows:

1. Constructing a benchmark library for function call sequences based on sliding windows. Traditional industrial control terminal program security protection methods often use signature- or rule-based detection methods, which have coarse detection granularity and cannot effectively detect and prevent attacks. We use a sliding window execution sequence measurement method to measure the sequence of application layer function calls, and establish a complete lightweight benchmark library based on the measurement values. The benchmark library lays the foundation for the subsequent implementation of the dynamic credibility evaluation of application layer function call sequences.
2. Proposing a partition dynamic credibility evaluation mechanism. Traditional industrial control terminal security detection technology usually uses offline scanning for detection, which lacks real-time performance and cannot respond quickly to attack events. Meanwhile, the system calls generated during program execution are unstable and difficult to measure. We propose a dynamic credibility evaluation method between partitions, which divides runtime program behavior into application layer function call behavior and system call behavior within function intervals. For the application layer function call sequence, a trust measurement method based on sliding window execution sequence is used. The real-time measurement results are evaluated

based on the benchmark library. For the system call sequence within the function interval, a maximum entropy system call model is constructed and used to evaluate the credibility of the system call sequence.

In Section 2, we review approaches in the existing literature. In Section 3, we introduce the mechanism of program behavior trust measurement, including the dynamic credibility measurement of function call sequence based on sliding windows and the credibility evaluation for system calls within the interval based on maximum entropy. In Section 4, we conduct experiments on the method proposed in this article and present the results. Section 5 concludes the paper.

2. Analysis of the Current State of Research

This article proposes a fine-grained dynamic credibility measurement and evaluation for industrial control terminal, which requires a trusted behavior measurement analysis of program behavior. Trusted behavior measurement is the process of analyzing program execution behavior to determine if the actual behavior matches the expected behavior. The program behavior integrity measurement framework IMA [8] employs a "load-time measurement" approach, which uses hash algorithms to verify whether the program code matches the expected behavior to ensure software behavior integrity. However, this method can only indicate that the program behavior has not been tampered with before running, and cannot verify whether the program behavior during running is trustworthy.

Control Flow Integrity (CFI) achieves trusted behavior measurement by restricting unexpected control flow to unauthorized positions [9]. Many CFI technologies have been proposed in recent years [10–12], mainly divided into two types: fine-grained and coarse-grained. Fine-grained CFI, also known as strict type CFI, has high accuracy and effectively identifies control flow attacks to ensure software behavior integrity while reducing false positives. However, since fine-grained CFI usually enforces shadow stacks, it results in significant performance overhead. In order to solve performance issues, some researchers proposed coarse-grained CFI, also known as loose CFI [13], which has lower performance overhead, but has been proven to be relatively less secure [14] and has certain limitations in obtaining application details [15].

Due to the issues of large performance overhead, complex implementation processes, and poor compatibility in measuring program behavior integrity, CFI cannot meet the requirements of trusted behavior measurement in industrial control scenarios. Abera et al. [16] proposed a Control Flow Attestation (C-FLAT) architecture for verifying the trustworthiness of program behavior on industrial control devices, but this method involves a great deal of computation and the classification processing of some basic blocks, resulting in low real-time performance.

Reference [17] creates a software behavior model based on n-gram [18,19], using fixed-length system call sequences to represent program behavior, and predicting the next system call sequence by statistically analyzing the frequency of n consecutive system calls. Since this model does not assume any prior probability distribution, it is sometimes more applicable than Markov chain and Hidden Markov Models. However, when using n-gram to create a program behavior model, the program behavior sequence needs to be divided into multiple sub-sequences of the same length, which is less flexible in the application environment. The variable-length sequence model (Var-gram) [20] has improved the n-gram model, but both models represent all system calls, including some low-security system calls, which increases the time cost of the program trusted behavior measurement process.

Dynamic symbolic execution [21–23] can integrate specific program execution behaviors and program symbolic execution behaviors, providing an accurate memory model and significant effect in detecting memory-related errors. However, symbolic execution is prone to path explosion problems and lacks scalability [24,25], making it unsuitable for industrial control scenarios.

Amer E et al. [26] established a behavior model based on API call sequences for the trusted behavior of API calls in industrial control scenarios, depicting the actual behav-

ior relationship between API functions for devices with more API calls. However, this model is only applicable to industrial control devices with more API calls and has limited applicability for devices with fewer API calls.

Based on the characteristics of the program in industrial control devices, this article divides program behavior into function call behavior and system call behavior by function-level interval partitioning, and only considers important system calls for system call behavior, designing a program behavior dynamic trust measurement and evaluation system with accuracy and real-time performance.

3. Research Content

3.1. Function Call Sequence Trust Measurement Evaluation

Due to the fact that there are fewer and more stable call sequences during program execution, this paper uses a partial matching method to measure function call sequences. Traditional measurement methods include one-to-one matching and overall matching. If one-to-one matching is performed for function call sequences, the time for dynamic trust measurement will be increased. If the entire function call sequence is to be measured, the measurement result can only be obtained after the program is executed, which does not meet the real-time requirements of dynamic trust measurement. Therefore, this paper uses a sliding window execution sequence measurement method to partially track and measure the dynamically monitored function call sequence, and compares the real-time measurement results with the benchmark library to obtain a reliable evaluation result for the internal function call sequence of the program. The dynamic trust measurement evaluation process of function call sequences is shown in Figure 1, and func1-func16 is an example of a sequence of function calls arranged in order.

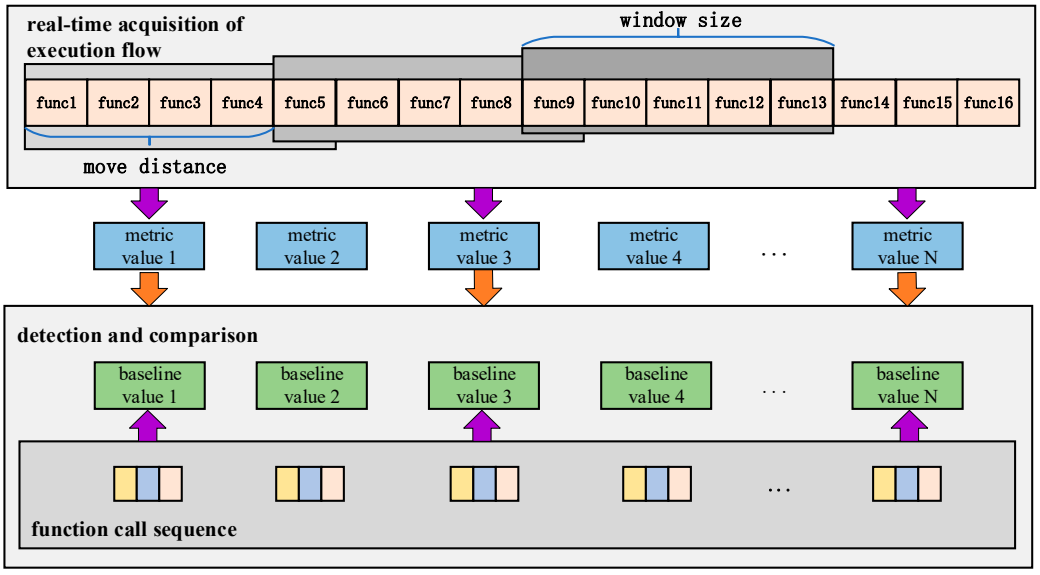

Figure 1. Schematic diagram of dynamic trust measurement evaluation process for function call sequences.

The dynamic trust measurement evaluation requires the use of a benchmark library as a comparison standard. Therefore, this article first constructs a low-complexity program execution flow graph, then traverses the program execution flow graph to obtain a complete sequence of function calls, and finally establishes a benchmark library through the sliding window execution sequence measurement method. The steps of this process are as follows.

3.1.1. Construction of Function-Level Execution Flow Graph (FEFG)

The function-level execution flow graph focuses on the control and data dependencies between functions, and is a prerequisite for extracting the sequence of function calls. Its construction process is shown in Figure 2.

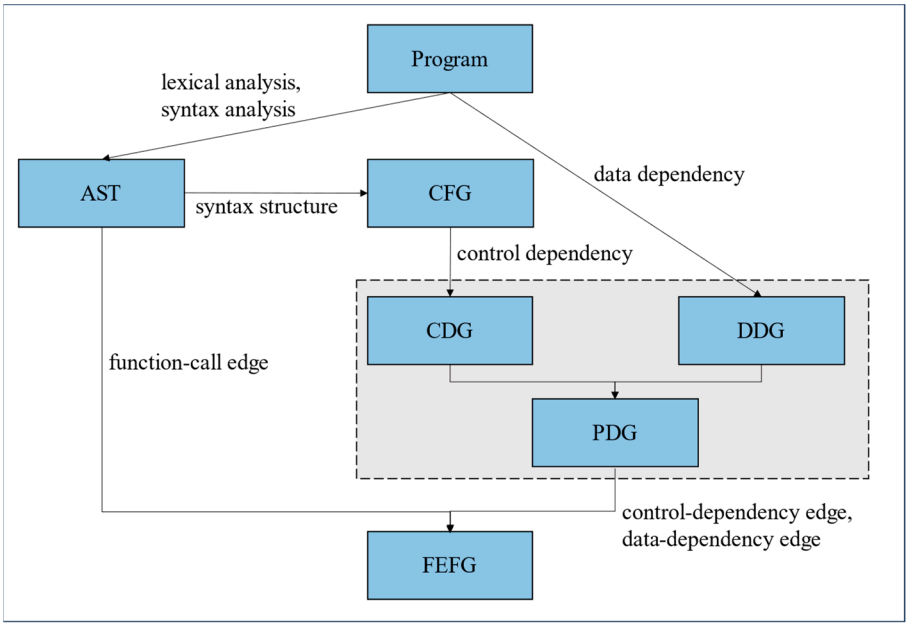

Figure 2. Construction process of function-level execution flow graph (FEFG).

1. Construction of Abstract Syntax Tree (AST)

Firstly, a lexical analyzer is used to scan the source program, recognize words based on word formation rules, generate lexemes, and ultimately generate lexical units as output. Next, a syntax analyzer is used to organize and convert the tokens' output by the lexical analyzer into sequences defined by the target language syntax, and construct tokens into hierarchical structures such as syntax analysis trees or abstract syntax trees.

2. Construction of Control Flow Graph (CFG)

A control flow graph expresses the possible flow directions of all basic blocks in a program, consisting of basic blocks and control flows, where basic blocks are nodes of the graph and control flows are directed edges of the graph.

Assuming the set of program basic blocks is $B = \{v_0, v_1, \ldots, v_n\}$, for each basic block v_{i}, there are corresponding start instructions In_{i} and stop instructions Out_{i}. By traversing the stop instruction of each basic block v_{i}, if the target of the stop instruction is to start the instruction of basic block v_{j}, a directed line segment is drawn from v_{i} to v_{j} as the control flow from basic block i to basic block j. After traversing all the basic blocks, the entire CFG is obtained. The basic blocks with an in-degree of 0 are the entries of the program, and the basic blocks with an out-degree of 0 are the exits of the program.

The construction algorithm of CFG is shown in Algorithm 1.

3. Construction of Program Dependency Graph (PDG)

The program dependency graph is obtained from the data dependency graph (DDG) and the control dependency graph (CDG), where the data dependency graph describes the constraint relationship between data, while the control dependency graph describes the constraint relationship of statement execution.

Firstly, Algorithm 2 is used on the basis of CFG to construct a CDG based on the forward dominance relationship of nodes.

Then, DDG is constructed based on all variables in the program and their relationships. The construction process is shown in Algorithm 3.

Algorithm 1: Program Control Flow Graph Construction Algorithm

Input: G_{ast} = (*V*, *E*, *root*), program source code
Output: G_{cfg} = (*V*, *E*, *entry*, *exit*)

```
def buildCFG(G_ast):
    Initialize G_cfg;
    for v in G_ast:
        if isConditionNode(v) then
            branch(v, G_cfg);
        if isSequentialNode(v) then
            next(v, G_cfg);
        if isReturnNode(v) then
            sign(v, G_cfg);
        if isJumpNode(v) then
            target(v, G_cfg);
    return G_cfg;
```

Algorithm 2: Control Dependency Graph Construction Algorithm

Input: G_{cfg} = (*V*, *E*, *entry*,*exit*)
Output: G_{cdg} = (*V*, *E*, *entry*)

```
def buildCDG(G_cfg):
    Initialize G_cdg;
    for v in G_cfg:
        for each subsequent node v_s of v:
            if there are no loops on the path from v to v_s:
                add_directed_edges(v_s, v, G_cdg);
            if the path from v to v_s contains one or more loops:
                for the head node v_l of innermost loop:
                    add_directed_edges(v_l, v, G_cdg);
        for each dominant subtree v_d of v:
            add_directed_edges(v_d, v, G_cdg);
    return G_cdg;
```

Algorithm 3: Data Dependency Graph Construction Algorithm

Input: Collection *N* of variables, constants, functions, and statement blocks
Output: G_{ddg} = (*V*, *E*, *var*)

```
def buildDDG(N):
    Initialize G_ddg
    for v in N:
        for each input variable v_x of v:
            if v_x not in G_ddg:
                add(v_x, G_ddg);
            for all statement nodes v_u before v:
                if the output variable of v_u contains v_x:
                    add_directed_edges(v_u, v_x, G_ddg);
        for each output variable v_y of v:
            if v_y not in G_ddg:
                add(v_y, G_ddg);
            for all statement nodes v_w after v:
                if the input variable of v_w contains v_y:
                    add_directed_edges(v_y, v_w, G_ddg);
    return G_ddg
```

Finally, the nodes in the CDG are traversed, which determine the basic relationship between each statement represented by the nodes. Then, data dependency edges and

control dependency edges are established based on the types of operands in the statement nodes. Finally, PDG is obtained. The merging process is shown in Algorithm 4.

Algorithm 4: Program Dependency Graph Construction Algorithm

Input: G_{cdg} = (*V*, *E*, *entry*), G_{ddg} = (*V*, *E*, *var*)
Output: G_{pdg} = (*V*, *E*, *entry*, *exit*)

```
def buildPDG(G_cdg, G_ddg):
    Initialize G_pdg;
    for v_B in G_cdg:
        add(v_B);
        for each statement I in v_B:
            add(I, G_pdg);
                for each operand O of I:
                    if O is variable:
                        add_data_dependency_edge(O, P, G_pdg);
                    if O is label:
                        add_control_denpendency_edge(O, v_B, G_pdg);
        for the out-edge E of v_B:
            if Q is a conditional branch or loop:
                add_control_denpendency_edge(v_B, O, G_pdg);
            if Q is an unconditional branch:
                add_control_denpendency_edge(v_B, O, G_pdg);
                For each successor R of Q and the successor S of v_B:
                    add_control_denpendency_edge(R, S, G_pdg);
    return G_pdg;
```

4. Construction of Function Level Execution Flow Graph (FEFG)

AST is generated based on functions, so we use AST to extract function calls. Then, the PDG is combined to increase control and data dependencies between functions. Finally, a complete low complexity program structure diagram is obtained. The construction process is shown in Algorithm 5.

Algorithm 5: Function-level Execution Flow Graph Construction Algorithm

Input: $G_{ast\text{-}unite}$ = (*V*, *E*, *root*), G_{pdg} = (*V*, *E*, *entry*,*exit*)
Output: G_{fefg} = (*V*, *E*, *entry*, *exit*)

```
def buildFEFG(G_ast-unite, G_pdg):
    Initialize G_fefg;
    for func in G_ast-unite:
        G_fefg [func] ← set()
    for func in G_pdg:
        for each successor succ of func:
            add_control_dependency_edge(G_fefg, func, succ);
        for each predecessor pred of func:
            add_data_dependency_edge(G_fefg, pred, func);
    return G_fefg;
```

3.1.2. Construction of Benchmark Library

The construction of a benchmark library relies on the sequence of function calls during program runtime. In order to determine the order of function calls and parameter transfer methods, the acquisition of function call sequences requires the static analysis of the function level execution flow graph (FEFG), and the reachability and data dependency relationships between functions. Therefore, this article uses depth first search (DFS) [27] to traverse FEFG and obtain the sequences of function calls.

After obtaining the function call sequences, a sliding window fast measurement mechanism is used to measure them partially, and the measurement results are stored as benchmark values in the benchmark library, laying the foundation for the implementation of dynamic credibility measurement evaluation in the program. The construction process of the benchmark library is shown in Algorithm 6.

Algorithm 6: Measurement Algorithm for Function Call Sequences Based on Sliding Windows

```
Input: func_call_seq
Output: hashValue
Set window_ size;
Set sliding_step;
Initialize an empty window list win_ list;
Initialize a temporary window temp_ win;
for each function func in func_call_seq:
    while size of temp_win is not equal to window_size:
        add func to temp_win;
    add temp_win to win_list;
    slide temp_win;
    repeat step7 to step9 until the func_call_ seq is traversed;
for each window w in win_list:
    add hash value of w to hashValue;
return hashValue;
```

3.1.3. Dynamic Credibility Measurement of Function Call Sequence Based on Sliding Windows

Quick measurements are carried out for each function call subsequence in each sliding window, and then the measurement results are traversed and matched in the benchmark library for security evaluation. If the match fails, the program behavior is considered untrustworthy. The program will be terminated and an error message will be issued. If the match succeeds, the address where the matching value is located in the benchmark library is recorded as the base address for subsequent comparison, and the sliding window continues to move forward. The function call sequence trust measurement evaluation algorithm is shown in Algorithm 7.

Algorithm 7: Trust Measurement Evaluation Algorithm for Function Call Sequences Based on Sliding Windows

```
Input: Benchmark library, real-time measurement value
Output: Evaluation result
while the program is running do
get the real-time execution flow of function calls;
slide window forward;
    generate metric-object using window;
    measure each metric-object;
    match metric-results in the benchmark library;
if match failed:
    terminate the program;
    return NOT_TRUSTED;
get address of metric-results;
continue program execution;
end while
return TRUSTED;
```

The storage address of the first measurement value in each function call path is used as the address of that path. During dynamic matching, if the current matching value is found in an execution path, then the function call sequence within the current window is part of that execution path. The address where the matching value is located in the benchmark

library is recorded, and used as the base address for subsequent comparison with the benchmark values that follow along that path. The subsequent function call subsequences will be compared within the subsequent windows along the path.

3.2. Trust Measurement Evaluation for System Call Sequences

Due to the uncertainty of program execution, a large and variable number of system call sequences are generated at each time the program runs. If a matching method is used to measure these system call sequences, a high false positive rate will occur. Therefore, the uncertainty of program system call sequence behavior needs to be taken into account. Since the maximum entropy is a probability statistical method used to solve uncertainty problems, this paper models the behavior of the function interval based on the maximum entropy principle, and computes the probability of the occurrence of key system call sequences. Based on the probability distribution, whether a new system call sequence is trustworthy can be judged, which effectively reduces the false positive rate and improves the accuracy of measurement.

Therefore, compared with machine learning-related program behavior models, we propose a maximum entropy-based system call model, which uses the function interval as the unit of partition to divide the program into intervals and narrow down the range of program behavior represented by the model. The features within each function interval are represented independently of each other, which can improve the effectiveness of the features and the model.

3.2.1. Maximum Entropy Model

Jaynes [28] proposed the maximum entropy theory in 1957, which is based on the basic principle of establishing a statistical calculation model on known information factors. By excluding the influence of unknown factors, the most uniform probability distribution of known facts is obtained, and the deviation is minimized while maintaining uncertainty. The maximum entropy theory achieves this goal by making optimal inferences about unknown factors under known conditions, where entropy is used to measure uncertainty, and maximum entropy corresponds to the prediction model under maximum uncertainty.

In the principle of maximum entropy, it is necessary to first define some characteristic functions $f_i(x,y)$, where x represents the input variable and y represents the output variable. These characteristic functions can be in any form, such as indicator functions, polynomial functions, etc.

Then, some constraints need to be given, that is, the expectation $E_{\overline{P}}(f_i)$ of the feature function $f_i(x,y)$ regarding the empirical distribution $\overline{P}(X,\ Y)$ is equal to the expectation $E_P(f_i)$ of the feature function $f_i(x,y)$ regarding the conditional distribution $P(Y|X)$ and empirical distribution $\overline{P}(X)$, as shown in Formulas (1)–(3).

$$E_{\overline{P}}(f_i) = E_P(f_i) \tag{1}$$

$$E_{\overline{P}}(f_i) = \sum_{x,\ y} \overline{P}(x,\ y) f_i(x,y) \tag{2}$$

$$E_P(f_i) = \sum_{x,\ y} \overline{P}(x,\ y) P(y|x) f_i(x,y) \tag{3}$$

Next, the objective function is defined, which is the conditional entropy on the conditional probability distribution $P(Y|X)$, as shown in Formula (4). This objective function can be understood as maximizing the uncertainty or entropy of the system.

$$H(P) = -\sum_{x,y} \overline{P}(x) P(y|x) log P(y|x) \tag{4}$$

The solution of the maximum entropy model can be achieved through the Lagrange Multiplier Method, as shown in Formulas (5) and (6).

$$L(P,\lambda) = H(P) - \sum_i \lambda_i \left(\sum_y P(y \mid x) f_i(x,y) - E[f_i(x,y)] \right) \tag{5}$$

Then, the derivative of $L(P,\lambda)$ over $P(y|x)$ can be found, and be made equal to 0, and then the optimal solution is obtained.

$$P(y|x) = \frac{1}{Z(x)} exp\left(\sum_i \lambda_i f_i(x,y) \right) \tag{6}$$

where $Z(x)$ is the normalization constant. The maximum entropy model is not unique; it depends on the selection of the feature function $f_i(x,y)$ and the setting of constraint conditions. Therefore, in practical applications, it is necessary to make reasonable feature selection and constraint conditions based on specific problems.

3.2.2. System Call Credibility Measurement Model Based on Maximum Entropy

In this paper, the maximum entropy system call model is applied to the dynamic credibility measurement evaluation of system calls within the function interval. The main process is as follows:

(1) Feature extraction. Select appropriate features according to different tasks and requirements, such as call frequency, call time, etc., and convert the original data into feature vectors.
(2) Model training. Use the maximum entropy model to train the model and obtain a model with high accuracy and strong generalization ability. Through training, the probability distribution is obtained.
(3) Prediction and evaluation. Use the existing eigenvectors to predict the results of unknown data, calculate its probability distribution, and evaluate the safety of the program.

Through the above processes, we design a system call model based on maximum entropy for system calls within function intervals. The architecture is shown in Figure 3, which is mainly divided into two parts:

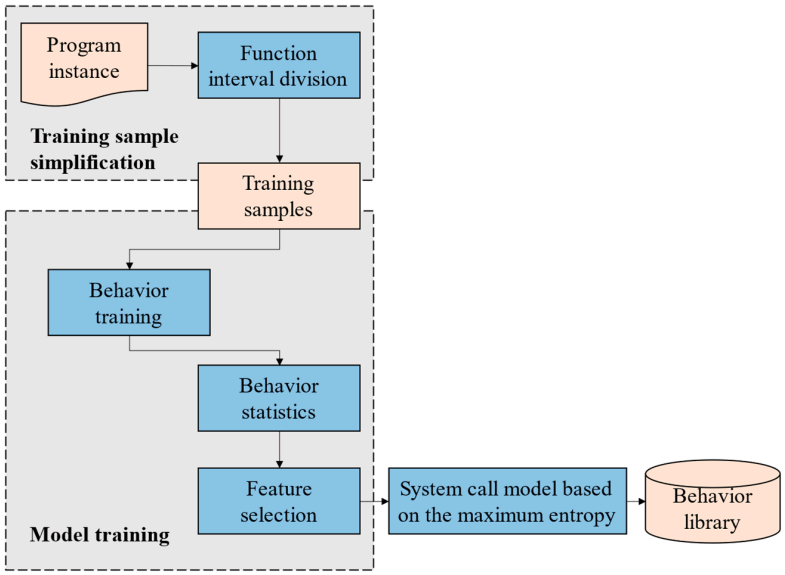

Figure 3. Architecture of interval system call model based on maximum entropy.

(1) Training sample simplification: It mainly simplifies the training data, divides the behavior intervals by function, obtains the key system calls and system calls with high security in the interval, and obtains the training samples of the maximum entropy model.

(2) Model training: The training system trains the training samples, counts the behavior probability, extracts the characteristics of the maximum entropy model, establishes the system call model in the interval based on the maximum entropy, and stores it in the behavior database.

Based on the above model architecture, this paper proposes an algorithm for constructing a system call model based on maximum entropy. The variables and symbol meanings used in Algorithm 8 for constructing a maximum entropy model are shown in Table 1.

Table 1. Variable and symbol meanings.

Variables and Symbols	Meanings
D	training dataset
x_i	input vector of the i-th sample
y_i	true output of the i-th sample
$f_j(x,y)$	a specific feature function that represents a feature function that satisfies the constraint condition $h_j(y)$ when the input is x and classified as y
$h_j(y)$	constraint function that represents the value of the j-th feature when the input is x and classified as y
w_j	weight of the j-th feature
$\varphi(x,y)$	feature vector of input vector x and classification y
$P(y \mid x)$	conditional probability of being classified as y given input x
$Z(x)$	normalization factor used to ensure that the sum of probability values equals 1
$E_p(h_j)$	expected value of feature function $h_j(y)$ under the current model
$E_s(h_j)$	expected value of feature function $h_j(y)$ given the training dataset D
λ	regularization parameter used to prevent overfitting

Based on the model architecture and the variable and symbol definitions in the above model, the maximum entropy system call model algorithm is designed according to Algorithm 8.

Algorithm 8: Construction Algorithm for the Maximum Entropy System Call Model

Input: training dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$
feature function $f(X) = \left\{f_1(X), f_2(X), ..., f_j(X)\right\}$
Output: the maximum entropy system call model $P(y \mid x)$

1. Set $h_j(y)$ and $f_j(x,y)$ based on $f(X)$
2. Initialize w;
3. for each sample $s(x, y)$:
4. $\varphi(x,y) \leftarrow convert\_to\_feature\_vector(s)$;
5. for each input x:
6. $Z(x) \leftarrow \sum_y exp\left[\sum_j w_j h_j(y)\right]$;
7. for each input x and its corresponding true value y:
8. $H(y \mid x) \leftarrow -\sum_y P(y|x) log P(y|x)$;
9. $H_s(Y) \leftarrow -\sum_{y \in Y} \left[\frac{1}{m}\sum_{i=1}^{m} I(y_i = y)\right] log\left[\frac{1}{m}\sum_{i=1}^{m} I(y_i = y)\right]$;
10. for each feature function $h_j(y)$:
11. $E_p(h_j) \leftarrow \sum_{x,y} p(x) P(y|x) h_j(y)$;
12. $E_s(h_j) \leftarrow \sum_{x,y \in D} h_j(y) p(x,y)$;
13. $w_j \leftarrow w_j + \frac{1}{\lambda} log \frac{E_s(h_j)}{E_p(h_j)}$;
14. Exit the iteration when the change in the weight vector is less than the threshold value.
15. $P(y \mid x) \leftarrow \frac{1}{Z(x)} exp\left[\sum_j w_j h_j(y)\right]$;
16. return the model $P(y \mid x)$;

To determine the conditional probability distribution of the maximum entropy system call model, the normalization factor, conditional entropy, empirical entropy, and expected value are required to be determined. The weight parameter λ of the feature function is then determined using the Improved Iterative Scaling (IIS) method [29]. The weight is iterated until it is smaller than the set threshold value. Then, the conditional probability distribution is computed. For each system call sequence, various features are extracted. After completing the feature extraction, a feature space based on the preprocessed data is constructed. For the first sequence, the feature space includes the frequency of four system calls, namely, open, read, write, and close, as well as the time interval and combination method between adjacent system calls. The model parameters are calculated, and then the maximum entropy algorithm is used based on the system call model within the interval to optimize the model parameters.

This article uses the maximum entropy system call sequence model to determine the credibility of program behavior within a function interval based on the probability of key system call sequences appearing. When measuring system call behavior within a function interval, behavior characteristics can be identified and extracted by collecting program behavior information during runtime. So the feature functions and constraint functions are determined, and the credibility of system calls are judged based on probability values, as shown in Figure 4.

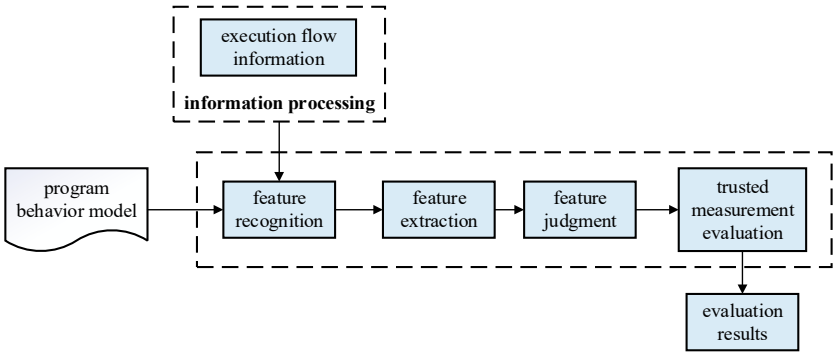

Figure 4. Process of credibility evaluation for system calls within the interval based on maximum entropy.

In the evaluation of system call credibility metrics within the function interval, the main process includes feature recognition, feature extraction, feature judgment, and trustworthiness metric evaluation. By analyzing and processing the system call behavior in the function interval, the credibility of program behavior can be accurately evaluated.

In the credibility measurement evaluation, the feature recognition phase will read the system call behavior template of the process in the program interval behavior library according to the number of the process to be measured, and screen the program-related features as alternative features. The template mainly contains information such as the division of function intervals, the behavior characteristics of system calls within function intervals, and the corresponding entropy value.

In the feature extraction phase, the credibility metric will select the behavior features of the process to be measured in this running process from the alternative features, and establish a feature list. The list includes two parts: feature representation and feature occurrence frequency.

In the feature determination phase, the trust measurement part classifies the features and determines whether the system call behavior in the interval is abnormal by judging the features. Specifically, the probability of the current feature is calculated to determine whether the probability exceeds the probability threshold. If the probability exceeds the threshold, it is considered that the feature determination fails; otherwise, the feature determination will continue.

In the credibility measurement evaluation stage, the credibility measurement evaluation module will collect the results of feature determination, and determine the final credibility measurement results of system call behavior in the current function interval based on the results of system call behavior determination in the function interval, so as to obtain the determination of program credibility.

Based on the above process, the maximum entropy-based system call credibility measurement evaluation algorithm designed in this article is shown in Algorithm 9.

Algorithm 9: Maximum Entropy-Based System Call Credibility Measurement Evaluation Algorithm

Input: the feature set $F = \{f_1(X), f_2(X), \ldots, f_n(X)\}$, the new function interval system call sequence F'
Output: the credibility measure *score* of F

for each feature $f_i(X)$:
$D_i \leftarrow$ *calculate_occurrence_frequency*();
$DS_i \leftarrow$ *calculate_standard_occurrence_frequency*();
for each feature $f_i(X)$:
$p_i \leftarrow D_i / |F|$;
$pS_i \leftarrow DS_i / |S|$;
$w_i \leftarrow log(p_i / pS_i)$;
define feature vector $X = (x_1, x_2, \ldots, x_n)$, x_i represents the frequency of the i-th feature in F'.
calculate the occurrence frequency of each feature in F' to obtain the vector x.
for F':
$H(F') \leftarrow -\sum_{i=1}^{n} p_i \times log p_i$;
$H_{\max} \leftarrow -\sum_{i=1}^{n} w_i \times x_i$;
$score \leftarrow 1 - (H(F')/H_{max})$;
return *score*;

4. Experimental Verification

In order to validate the method proposed in this article, we designed and implemented a program behavior dynamic credibility measurement evaluation system, which mainly includes a behavior dynamic monitoring module, a behavior organization module, and a credibility measurement evaluation module. Among them, the dynamic monitoring module obtains program behavior call information during program execution. The behavior organization module converts the program behavior information obtained from the behavior dynamic monitoring module into the format required by the credibility measurement evaluation module. The credibility measurement module is a key module for measuring program behavior during dynamic runtime, which is responsible for evaluating the credibility of application layer function call behavior information and system call behavior information. By testing the effectiveness and performance of the model, and comparing it with the current influential C-FLAT scheme in the field of trusted measurement in industrial control scenarios for attack detection, we analyzed the practical value of this system.

4.1. Evaluation of Model Effectiveness

This article evaluates the effectiveness of the model by measuring the error between predicted and true values. Specifically, two indicators are used: Mean Absolute Deviation (MAD) and Mean Absolute Percentage Error (MAPE). MAD is an indicator that measures the average absolute error between the predicted and actual values of a behavioral model, and can effectively reflect the accuracy of the model. MAPE is used to represent the unbiasedness of behavior. In the process of establishing a behavioral model, both of these indicators are of great significance and can help evaluate the accuracy and reliability of the model. The calculation method is shown in Formulas (7) and (8).

$$MAD = \frac{1}{N}\sum_{t=1}^{s} e_t \tag{7}$$

$$MAPE = \frac{1}{N}\sum_{t=1}^{s}\left|\frac{e_t}{d_t}\right|(\times 100\%) \tag{8}$$

where e_t is the prediction error of state t, $e_t = d_t - \overline{d_t}$. d_t is the predicted value of state t, $\overline{d_t}$ is the actual value of state t, and N is the total number of times the experiment was conducted.

This article uses the experimental sample database VX Heaven [30] to test the effectiveness of the Var-gram model [31], Dyck model [32], FSA model [31], and our model. The Var-gram model is a text generation model based on n-grams, which uses a variable length context window to integrate information from multiple windows through weighted averaging. The Dyck model is a text classification model based on a sequence of parentheses, which treats various parentheses in the text as nodes and constructs a tree structure to represent the syntactic structure of the text. The FSA model is a formal language computing model that consists of three important elements: a set of states, a transition function, and a set of accepting states. The state set includes all possible states, the transition function defines the transition rules between states, and the accepting state set indicates which states are terminating or acceptable.

This article uses generative model to learn from sample data and ultimately generates behavioral data for experimentation. The state monitoring number l is set. The sequence differences are compared between the behavior sequence obtained by the expected behavior model and the benchmark sequence of the sample. And the accuracy and reliability of the behavior model can be evaluated. In this article, N is set to 20 and l is set to 1000, which means setting up 1000 random collection points and conducting 20 experiments. Firstly, the MAD values in two scenarios are compared, and s states from 1000 states are extracted as program behavior state detection points to collect model information ($20 \leq s \leq 1100$). The comparative experimental results of MAD under different monitoring states are shown in Figure 5.

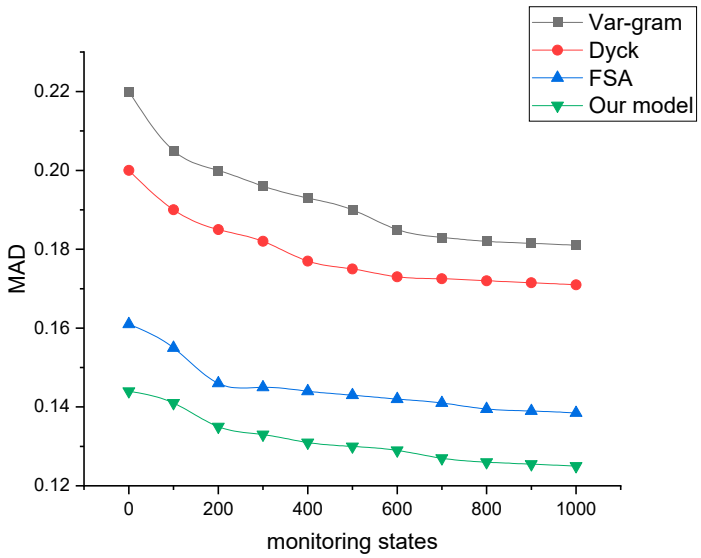

Figure 5. Comparison of MAD under different monitoring states.

According to the results, it can be seen that when the number of state detection points s is small, the MAD value of our model is significantly better than the other three models. For example, when s = 20, the MAD value of our model is 10.49% lower than that of the FSA model. When s > 300, the curve of our model is relatively flat, with MAD values tending towards 0.125 and consistently lower than FSA's 0.14, indicating that our model

has better accuracy. When $s > 600$, the two curves of our model and the FSA model gradually approach, indicating that when the obtained program behavior states reach a certain number, the performance of the model tends to stabilize.

Similarly, s states were selected from 1000 states to collect information from the model, with $20 \leq s \leq 1100$. A total of 20 calculations were conducted to verify the unbiased nature of the model. The Var gram model, FSA model, and Dyck model were compared. The experimental results of the MAPE comparison under different monitoring states are shown in Figure 6.

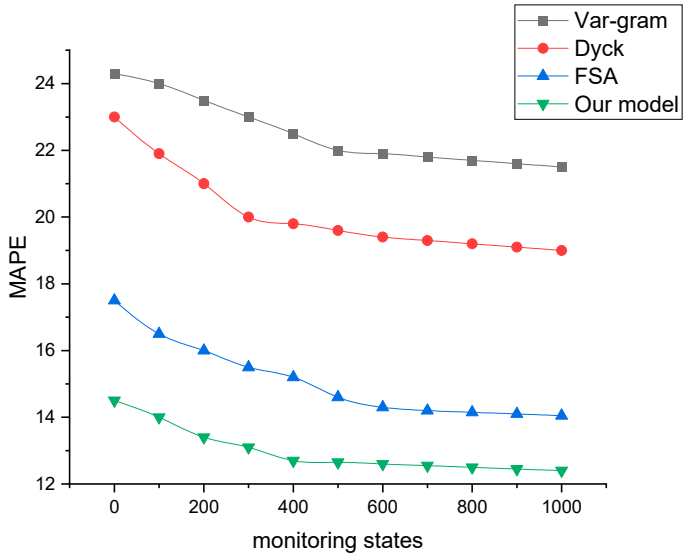

Figure 6. Comparison of MAPE under different monitoring states.

The comparison results of MAPE indicate that our model tends to flatten compared to the FSA model, and the MAPE value of our model is 17.14% lower than that of the FSA model. When n > 400, the MAPE values of our model are all less than 13 and gradually tend to 12, while the MAPE values of the other models are all greater than 14. Therefore, our model has better unbiasedness.

4.2. Comparative Experiment on Attack Detection

This section compares the dynamic credibility measurement mechanism based on function partition in this paper with the C-FLAT scheme [24], which has certain influence in the field of trust measurement in industrial control scenarios, and analyzes its practical value.

This experiment uses the sample database VX Heaven [30] to compare the detection of three types of attacks, namely, ROP attack, mimicry attack, and code injection attack, by both our system and the C-FLAT method, and records the time when the two methods detect the attacks separately. This paper downloads various types of malicious software samples from the sample library, modifies some code according to the attack method, and compiles and runs the malicious code for testing. A ROP attack constructs ROP attack code in the sample using techniques such as memory overlay and pointer replacement. A mimicry attack involves adding or replacing instruction parameters and exchanging instruction order in the code instruction sequence. A code injection attack changes conditional judgments, loop structures, or jump statements in the code.

The experiments show that our system and the C-FLAT method can detect both ROP attacks and code injection attacks. As for mimicry attacks, the C-FLAT scheme can partially detect them, but cannot completely prevent this type of attack. That is because after the program is mutated by a mimicry attack, some malicious codes' control flow still conforms to the detection rules of C-FLAT, which can bypass the detection of C-FLAT. Our method separates function call flow and system call flow for detection, and considers both control

flow and data flow during the calling process. Therefore, our method can effectively prevent mimicry attacks compared to the C-FLAT.

The performance test results are shown in Figure 7.

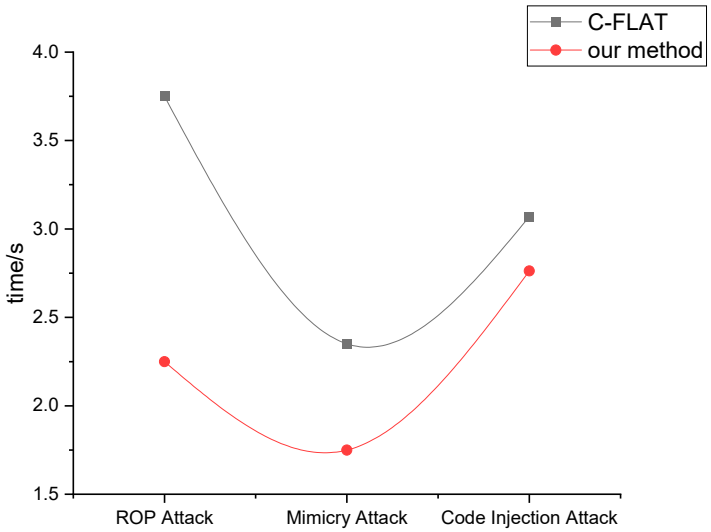

Figure 7. Comparison results with C-FLAT.

The experimental results show that in the detection of ROP attacks, our system's detection efficiency is 40% higher than that of the C-FLAT method. In the detection of mimicry attacks, our system's detection efficiency is 48% higher than that of the C-FLAT method. In the detection of code injection attacks, our system's detection efficiency is 10% higher than that of the C-FLAT method. That is due to the sliding window execution sequence measurement method used in this paper, combined with the benchmark library for comparative evaluation, which improves the timeliness of attack detection. Based on the maximum entropy system call model, the program behavior is evaluated by probability distribution, which can reduce the classification discussion of different system calls and reduce the time for credibility measurement evaluation. Therefore, our system's detection efficiency for these three types of attacks is higher than that of the C-FLAT method using full path measurement.

4.3. Evaluation of Model Performance

In dynamic credibility measurement, it is necessary to perform a series of operations on the program being measured during runtime, such as dynamic collection, sequence processing, baseline querying, and credibility evaluation. These operations will bring latency to the system operation, and the level of latency directly affects whether the system can be used in industrial control scenarios. In order to test the performance of the system, this article sets timers on the key components of the dynamic credibility evaluation prototype system to calculate the average execution delay of each measurement stage. Under two schemes based on n-gram [33] and our method, 1000 dynamic credibility measures were performed on the same sample program, and the total average latency of the credibility evaluation system was obtained. The test results are shown in Figure 8.

Through the analysis of experimental data, it can be seen that there are differences in the impact of credibility measurement evaluation systems based on different behavioral models on latency. During the dynamic collection process, the n-gram model and our model take almost the same amount of time. In the process of behavioral organization, due to the requirement of filtering the system call sequence, our model takes more time than the n-gram model. In the benchmark query process, due to the use of the sliding window execution sequence measurement method (window size set to 6), the window moves a distance of 5 per unit time, while the n-gram model based scheme moves a distance of 1 per

unit time, so our scheme is faster. In the credibility evaluation process, due to the use of an entropy-based system call model to measure and evaluate the key system call sequences within the function interval, the time used is lower than that of the n-gram model. Finally, by calculating the total latency of the two schemes, it can be seen that our system has a lower latency throughout the dynamic credibility evaluation process, indicating that this model has greater potential for the design of efficient credibility measurement schemes.

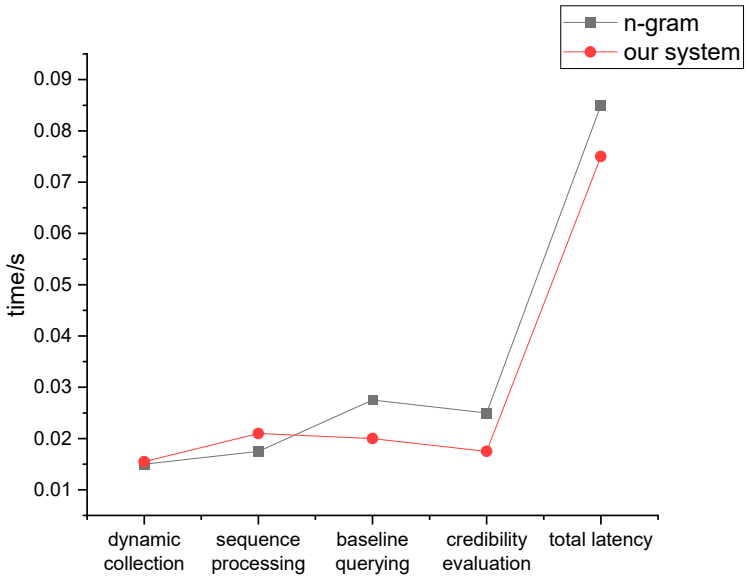

Figure 8. Comparison of dynamic credibility measurement latency.

4.4. Discussion

Firstly, this article uses the maximum entropy system call model for the dynamic trustworthiness evaluation of system call sequences. Therefore, it is necessary to demonstrate the effectiveness of the maximum entropy system call model through model accuracy experiments. The experimental results of this article indicate that due to the Var-gram model integrating information from multiple variable length windows through weighted averaging, the information from a single window has a greater impact on the integrated information, resulting in lower accuracy and unbiasedness of the model. The parentheses in the text information of the Dyck model are nodes, and the node information is relatively single, so its model accuracy and unbiasedness are also low. The FSA model considers each program state and aggregates the states, using a transition function to represent the transition of program states. Therefore, it has higher accuracy and unbiasedness than the Var gram model and the Dyck model. However, the FSA model requires a transition function to calculate and judge the states, which is highly dependent on the transition function. Our proposed model aims to make system calls and uses entropy to calculate the frequency of system calls. Based on the probability distribution, the system call sequence is classified. Therefore, compared to other models, our model has higher accuracy and unbiasedness.

Additionally, due to the use of the sliding window execution sequence measurement method, combined with a benchmark library for comparison evaluation, the timeliness of detecting attacks is improved. Based on the maximum entropy system call model, program behavior is further evaluated through probability distribution, which reduces the classification discussion of different system calls and reduces the time for trust metric evaluation. Therefore, the detection efficiency of our system for these three types of attacks is higher than that of the full path metric C-FLAT method.

5. Conclusions

Traditional security protection methods for industrial control terminals have coarse detection granularity, and are unable to effectively detect and prevent attacks, lacking

real-time responsiveness to attack events. Therefore, this paper proposes a mechanism of program behavior trust measurement that includes a sliding window execution sequence measurement method and a maximum entropy system call model based on the research of the static analysis and dynamic measurement. Contrast experiments with a typical industrial control terminal security protection technology C-FLAT scheme prove that our system can detect malicious attacks more accurately and more quickly, and is more suitable for terminal security protection in industrial control scenarios.

This article proposes a dynamic trustworthiness evaluation mechanism based on program behavior, which mainly improves on code reuse attacks and mimetic attacks in industrial control terminal applications. However, there are still some areas that can be improved. First, the dynamic credibility evaluation mechanism proposed in this article is based on the Linux operating system and provides an implementation method for dynamically monitoring system call sequences in the Linux operating system. The next step is to consider implementing it in multiple operating systems to enhance the portability of this mechanism. In addition, the maximum entropy-based system call model proposed in this article requires multiple iterations of historical data to obtain the optimal behavioral model. This process is relatively time-consuming and requires high data volume. Further optimization of the model is needed to improve its self-learning efficiency in the future.

Author Contributions: Conceptualization, S.W.; methodology, S.W., A.H. and S.L.; software, S.W., T.L. and S.L.; writing—original draft preparation, S.W. and S.L.; writing—review and editing, A.H.; visualization, T.L.; supervision, A.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Fundamental Research Funds for the Central Universities (No. 2242022k60005), Purple Mountain Laboratories for Network and Communication Security, and National Science Foundation (No. 62233003).

Data Availability Statement: The dataset presented in this study is available on https://github.com/opsxcq/mirror-vxheaven.org (accessed on 16 January 2024); For any other questions, please contact the corresponding author of this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tsochev, G.; Trifonov, R.; Nakov, O.; Manolov, S.; Pavlova, G. Cyber security: Threats and Challenges. In Proceedings of the 2020 International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 1–3 October 2020.
2. Ani, U.P.D.; Watson, J.M.; Green, B.; Craggs, B.; Nurse, J.R.C. Design considerations for building credible security testbeds: Perspectives from industrial control system use cases. *J. Cyber Secur. Technol.* **2021**, *5*, 71–119. [CrossRef]
3. Zhang, L.; Meng, Y.; Yu, J.; Xiang, C.; Falk, B.; Zhu, H. Voiceprint Mimicry Attack Towards Speaker Verification System in Smart Home. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications, Toronto, ON, Canada, 6–9 July 2020.
4. Luo, B.; Xiang, F.; Sun, Z.; Yao, Y. BLE neighbor discovery parameter configuration for IoT applications. *IEEE Access* **2019**, *7*, 54097–54105. [CrossRef]
5. Khedker, U.; Sanyal, A.; Sathe, B. *Data Flow Analysis: Theory and Practice*; CRC Press: Boca Raton, FL, USA, 2017; pp. 59–99.
6. Aghakhani, H.; Gritti, F.; Mecca, F.; Lindorfer, M.; Ortolani, S.; Balzarotti, D.; Vigna, G.; Kruegel, C. When malware is packin' heat: Limits of machine learning classifiers based on static analysis features. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2020, San Diego, CA, USA, 23–26 February 2020.
7. Shestakov, A.L. Dynamic measuring methods: A review. *Acta IMEKO.* **2019**, *8*, 64–76. [CrossRef]
8. Sailer, R.; Zhang, X.; Jaeger, T.; van Doorn, L. Design and implementation of a TCG-based integrity measurement architecture. In Proceedings of the 13th USENIX Security Symposium 2004, San Diego, CA, USA, 9–13 August 2004; pp. 223–238.
9. Koruyeh, E.M.; Shirazi, S.H.A.; Khasawneh, K.N.; Song, C.; Abu-Ghazaleh, N. Speccfi: Mitigating spectre attacks using CFI informed speculation. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 39–53.
10. Jeong, S.; Hwang, J.; Kwon, H.; Shin, D. A CFI countermeasure against GOT overwrite attacks. *IEEE Access* **2020**, *8*, 36267–36280. [CrossRef]
11. Feng, L.; Huang, J.; Hu, J.; Reddy, A. FastCFI: Real-time control-flow integrity using FPGA without code instrumentation. *ACM Trans. Des. Autom. Electron. Syst. TODAES* **2021**, *26*, 1–39. [CrossRef]

12. Serra, G.; Fara, P.; Cicero, G.; Restuccia, F.; Biondi, A. PAC-PL: Enabling control-flow integrity with pointer authentication in FPGA SoC platforms. In Proceedings of the 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS), Milano, Italy, 3 May 2022; pp. 241–253.
13. She, C.; Chen, L.; Shi, G. TFCFI: Transparent Forward Fine-grained Control-Flow Integrity Protection. In Proceedings of the 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Wuhan, China, 28–30 October 2022; pp. 407–414.
14. Moghadam, V.E.; Prinetto, P.; Roascio, G. Real-Time Control-Flow Integrity for Multicore Mixed-Criticality IoT Systems. In Proceedings of the 2022 IEEE European Test Symposium (ETS), Barcelona, Spain, 23–27 May 2022; pp. 1–4.
15. Li, Y.; Wang, M.; Zhang, C.; Chen, X.; Yang, S.; Liu, Y. Finding cracks in shields: On the security of control flow integrity mechanisms. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 1821–1835.
16. Abera, T.; Asokan, N.; Davi, L.; Ekberg, J.-E.; Nyman, T.; Paverd, A.; Sadeghi, A.R.; Tsudik, G. C-FLAT: Control-flow attestation for embedded systems software. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security 2016. Vienna, Austria, 24–28 October 2016; pp. 743–754.
17. Hu, H.; Shinde, S.; Adrian, S.; Chua, Z.L.; Saxena, P.; Liang, Z. Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P), San Jose, CA, USA, 23–25 May 2016.
18. Canonical. *Ubuntu Core—The Operating System Optimized for IoT and Edge*; Canonical: Eatontown, NJ, USA, 2022.
19. Werner, M.; Unterluggauer, T.; Schaffenrath, D.; Mangard, S. Sponge-Based Control-Flow Protection for IoT Devices. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 24–26 April 2018.
20. Shahzad, R.K. Android malware detection using feature fusion and artificial data. In Proceedings of the 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Athens, Greece, 12–15 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 702–709.
21. Cadar, C.; Sen, K. Symbolic execution for software testing: Three decades later. *Commun. ACM* **2013**, *56*, 82–90. [CrossRef]
22. Vishnyakov, A.; Fedotov, A.; Kuts, D.; Novikov, A.; Parygina, D.; Kobrin, E.; Logunova, V.; Belecky, P.; Kurmangaleev, S. Sydr: Cutting edge dynamic symbolic execution. In Proceedings of the 2020 Ivannikov ISPRAS Open Conference (ISPRAS), Moscow, Russia, 10–11 December 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 46–54.
23. Cadar, C.; Nowack, M. KLEE symbolic execution engine in 2019. *Int. J. Softw. Tools Technol. Transf.* **2021**, *23*, 867–870. [CrossRef]
24. Trabish, D.; Kapus, T.; Rinetzky, N.; Cadar, C. Past-sensitive pointer analysis for symbolic execution. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering 2020, Virtual, 8–13 November 2020; pp. 197–208.
25. Poeplau, S.; Francillon, A. Symbolic execution with SymCC: Don't interpret, compile! In Proceedings of the 29th USENIX Conference on Security Symposium 2020, Boston, MA, USA, 12–14 August 2020; pp. 181–198.
26. Amer, E.; Zelinka, I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* **2020**, *92*, 101760. [CrossRef]
27. Moore, E.F. The Shortest Path Through a Maze. In *Proceedings of the International Symposium on the Theory of Switching*; Harvard University Press: Cambridge, MA, USA, 1959.
28. Jaynes, E.T. Information theory and statistical mechanics. *Phys. Rev.* **1957**, *106*, 620. [CrossRef]
29. Berger, A.L. *The Improved Iterative Scaling Algorithm: A Gentle Introduction*; CMU School of Computer Science: Pittsburgh, PA, USA, 1997.
30. Vxheaven. Org's Website Mirror [EB/OL]. *(2018–07–28).* Available online: https://github.com/opsxcq/mirror-vxheaven.org (accessed on 20 December 2023).
31. Lai, Y.; Liu, Z.; Ye, T. Software behaviour analysis method based on behaviour template. *Int. J. Simul. Process Model.* **2018**, *13*, 126–134. [CrossRef]
32. Chen, X.; Ding, H.; Fang, S.; Li, Z.; He, X. A Defect Detection Technology Based on Software Behavior Decision Tree. In Proceedings of the 2017 International Conference on Computer Systems, Electronics and Control (ICCSEC), Dalian, China, 25–27 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 717–724.
33. Xiao, X.; Zhang, S.; Mercaldo, F.; Hu, G.; Sangaiah, A.K. Android malware detection based on system call sequences and LSTM. *Multimed. Tools Appl.* **2019**, *78*, 3979–3999. [CrossRef]