Protection Tiers and Their Applications for Evaluating Untrusted Code on A Linux-Based Web Server

Zhuhan Jiang, Jiansheng Huang, and Rezina Akhter
School of Computing, Engineering and Mathematics, University of Western Sydney, NSW, Australia
Email: {z.jiang, j.huang, r.akhter}@uws.edu.au

Abstract—Evaluating untrusted computer programs online by executing and testing them real-time has a challenging task of protecting the system integrity and the data confidentiality of the computer host. For the web based services on one of the most popular computer platforms, Linux, we propose three security protection tiers of different complexity and resource cost to incorporate the potentially unsafe application service via a web server. By utilizing a single regular Linux account of a corporate computer, or dual accounts as a combination, or multi-accounts of a dedicated computer, these three protection tiers can offer a trade-off between the simplicity in design and maintenance at the expense of a somewhat reduced security strength, and the more costly implementation and maintenance with a relatively better security strength. The need for such different tiers is especially true for our implemented in-house applications that aim to evaluate programming work automatically by executing the pertinent but untrusted client programs.

Index Terms—Execution protection strategies, execute unsafe programs, program code evaluation, convenience protection trade-off

I. INTRODUCTION

Executing non-trustworthy programs on any computer platform poses a grave security risk to the platform. Depending on the design of the operating systems, the risks can range from impacting an individual user account to destroying the whole computer system. There have been a great many efforts [1]-[3], to name a few, to minimize in general the security risks to maintain the system integrity or data confidentiality. However, it is well understood that it is in principle undeterminable whether an arbitrary program contains a computer virus, and in fact deliberate attacks can even lurk within a regular web advertising [4]. Nevertheless a great many security vulnerabilities that are published on various security mailing lists can be detected by some form of the fuzzing methods [5]. To counter the problems of untrusted or malicious code, sandboxing [6], [7] that originated from Java applet can come to our rescue for a range of scenarios. The sandbox-based approaches generally monitor a program’s execution behavior and block the actions that may compromise the system’s security. Such approaches may often have to incorporate highly restrictive security policies that could exclude many of the useful applications. Hence a more wholistic approach in the form of virtual execution environment or virtualization techniques has been widely adopted in the recent times, see e.g. [3], [8]. Of course, sandboxing and virtualization are not the only valid techniques [9] and neither can they address all execution protections. Hence for our purpose of executing and evaluating untrusted program code on a Linux based web server, we will concentrate on utilizing a variety of component techniques to achieve both the sufficient security protection and the desired operational simplicity, in a way that seems also uncannily reminiscent of the virtualization techniques we mentioned earlier on.

While virus scans, firewalls and user authentications can go a long way to avoid getting malicious programs such as computer viruses and spyware to sneak into a computer system, the most popular and easiest defense in this regard is not to execute any untrustworthy programs in this first place. However, this may be fundamentally impossible at times due to the nature of the circumstances or the application needs. For instance, a programming training course may need to evaluate a student client’s program by executing it first, and a security check at the level of source code analysis is typically inconclusive due to the lack of sophisticated artificial intelligence. The safest strategy is perhaps to set up a separate computer machine dedicated to executing such dangerous programs, and have the computer reformatted or reinstalled once all the risky executions are completed. However this is obviously very ineffective and often unnecessary, as a virtual environment [3] typically will also suffice. Hence a natural way to limit the risks or potential damages of executing an unsafe program is to limit its impact to the space of just one user account. If one standard user account under Linux has been messed up by certain application programs, then just delete the account and create a new one. On some platforms, it’s also possible to make use of security policies to set up a sandbox with certain features or capacities disabled. This nevertheless may reduce the capacity or the features of a programming language or environment, crippling the idea of executing a reasonably full-fledged application program. In general, a valid security measure or strategy has to match the application task in terms of the real-time feasibility, resource manageability and cost effectiveness. In this work, we will elaborate more on the protection tiers of...
employing merely one or two dedicated user accounts of the same server computer, regardless whether this computer is completely allocated for this purpose, even though multi-accounts on a dedicated server computer will also be addressed.

More precisely, our main purpose in this work is to propose different protection tiers so that they can sufficiently cater for the different level of security and convenience in regard to the execution of insecure program code. The stress is laid on the balance of the acceptable security with the desirable convenience, since a universal scheme of highest security and highest maintenance may make the system not implementable in the end. One of the applications for our proposed security tiers can be an automatic marking system that will be able to assess student programming exercises and assignments. An automatic assessment [10]-[12] of a piece of program code in terms of its functionality and performance can relieve the teaching staff of tremendous amount of tedious marking time, provide across the board assessment consistency and feedback, and to make possible the automatic programming drills or quizzes [13], [14]. There has been some literature on the design of automatic program marking for the selected platforms and programming languages [15], [16], most of which overlooked the security issues or simply assume the good intention of the student programs. One of our goals is thus to propose a program execution management of different level of security and maintainability, ranging from single-user or dual-user-account scheme, to the dynamic user creation and management, and to the exportation of the execution to the client machine, to name a few.

This paper is organized as follows. We will first in Section 2 propose a broad design for benchmarking potentially unsafe program source code, taking into consideration of the full security factors. We then in Section 3 address 3 different protection tiers on the application and web server according to the different available resources and the preferred tradeoffs. Some pertinent technical strategies, such as the execution delegation to client computers, for the security and other design aspects are then scrutinized in Section 4 to illustrate their application prospects. Section 5 then puts everything together for a concrete application, an online programming exercise and auto-marking system, develops the implemental interface, and illustrates the impact of utilizing different protection tiers. Finally Section 6 gives the conclusions.

II. OVERALL CONSIDERATIONS

In order to best identify the security issues surrounding the compilation and the execution of potentially dangerous code, we first explore the main strategies that can be employed to assess a given piece of program source code according to whether it has fulfilled the design requirement, and to what extent. By a program code in this work, we always assume that it is the source code in a particular programming language.

A. Benchmark Program Functionality

The evaluation of a program is often of two fold. One is about how well structured or how efficient the program is, and the other is about how completely the functionality has been achieved according to the specifications. The former is largely about the syntactic and semantic analysis on the source code, and the analysis itself has no impact on the security matters. While it constitutes a main research direction itself, it is not of our major concern. The analysis can range from the direct template match on the program structure to the more involved comparisons on the derived structure trees out of certain semantics. The latter is what we will look into in much more detail in this subsection.

By a program source code, we here refer to either a complete program which can be directly compiled and executed, or a program segment or even a single statement which will need to be combined with other program code before it can be compiled and executed. Let a program source be denoted by \( P = \{ P_i \}_{i \in I} \), where \( P_i \) represents a portion of the source code. \( P_i \) are assumed to be extractable automatically, and what \( P_i \) should be depends on the functionalities that need to be evaluated. In the case of a full source program \( P_{\text{full}} \) one can at least set \( I = \{ 1 \} \) and \( P = \{ P_0 \} \). The K-th functionality \( F^{(k)} \) for the program \( P \) can be described as

\[
O^{(k)} = A^{(k)} \left\{ \{ P_i \}, \{ Q_j \}, F^{(k)} \right\}
\]

implying that when the user code \( \{ P_i \} \) and the support code \( \{ Q_j \} \) are incorporated or combined according to an algorithm \( A^{(k)} \), the output \( O^{(k)} \) will lead to the output \( O^{(k)} \). Obviously, these \( \{ Q_j \}, A^{(k)}, F^{(k)} \), and \( O^{(k)} \) are all designed to evaluate or verify the anticipated functionality \( F^{(k)} \). In general, the functionalities \( F^{(k)} \) for \( k=0, ..., K \), are ordered in priority so that the later ones are expected to be evaluated after the completion of the earlier ones. In particular, \( F^{(0)} \) might typically denote that the compilation of the program is successful. If the evaluation algorithm \( A^{(k)} \) is executed on the server machine, then it becomes a security concern due to the use of untrusted code fragments. Unless the execution is delegated to the client machine somehow, the security issues will arise on the server.

B. Main Security Factors for the Server

Our main security factors are concerned with the unauthorized alteration of the system parameters, modification or theft of user data files, and the deterioration or deprivation of system services. For a server on a dedicated user account or computer, the unauthorized modification is of the highest concern, and then comes the data theft, and finally the erosion of the service quality.

The erosion of service quality is in general of no serious concern as most of such problems require the continuous execution of the culprit program, and this can
be essentially disabled by design as all the evaluation execution of the test program may be limited to only a given short period of time. This restriction is also typically activated to avoid the execution of infinite loops or the like. The theft of sensitive data is also of less significance, as it’s relatively easy to ensure that the current user account doesn’t archive any top confidential information, and all sensitive data for storage during the execution of the evaluation algorithms can be permanently exported to a different user account for instance. This leads us to focus more on the crucial security concern, the unauthorized modification of data or system parameters of the host account, which may as a result corrupt or even crash the server.

A natural question at this stage is to ask what program constructs in the source code could be responsible for modifying data or misusing resources. There are in fact two categories for such constructs. The first category is the constructs for shell into the operating system. Almost all modern programming languages support one or more such constructs. For C++, for instance, this could be in the form of `system()` or `exec()` series. The second category is the language constructs for file reading and writing. Monitoring the files to ensure they reside in a designated location, or even disabling file access completely unless absolutely necessary, will not only remove the risks of file modification but also further reduce the access of any sensitive data. For the input and output to evaluate a program, it’s safer to just use the standard input and standard output exclusively. This way the access to IO will be completely controlled by the design of the evaluation algorithm. For programming languages such as Java where security policies can be administered for each individual execution, forbidding unauthorized data access or shell can be much simpler though.

![Fig. 1. Controlled execution/evaluation of a user source program on the server](image)

**Fig. 1.** Controlled execution/evaluation of a user source program on the server

For all realistic purposes of executing generally small to medium student code, one can adopt different levels of security mechanism to protect the server from unintentional to even malicious damages. The securest approach would be to reload a completely refreshed operating system environment, have the testing code loaded, executed and evaluated, before reloading once again the untouched OS for testing the next program. This would however be too laborious to be really practical. Hence an acceptable trade-off between the level of security and the extent of convenience is what one needs to explore. In this regard, we will propose three protection tiers for the choices, with increased security and therefore the resulting decreased convenience.

At the lowest level of security protection, still quite acceptable for variety types of code testing, one can just make use of a single regular Linux account and utilise `cgiwrap` to execute all code with the account user’s access right. This requires the least of resources and is the easiest to implement. The main drawback of this approach is that the security check must be carried purely at the stage of the source code scrutiny. If any serious harmful bugs slip through this scrutiny and gets executed, then the worst is to delete the old account and recreate a new one, before batch installing all the additional software components. In fact any ordinary account can host such a service if sensitive data for other purposes are further protected by encryption or by backups elsewhere.

Although the whole system design is inclined for simplicity towards using merely one standard user account on a Linux machine, it will be further strengthened if two standard user accounts, a major and a minor, are utilized. This dual-account design will allow the application server to relax the restriction on for instance the explicit use of data files. The minor user account may even be routinely reloaded with a standard build in the backup to clear off any potential “damages” it may have received in the past, if certain less secure portion of the service is deliberately enabled to increase the scope and flexibility.

More specifically, this proposed dual-account server design will leave most of the server task to the user account named major, which will manage the client passwords and the client login, and serve all the static pages and the normal (secure) dynamic script pages. The user account minor will host the security-sensitive program evaluation system, utilizing the client credentials...

---

©2015 Journal of Communications
stored in the account major. Because these two accounts are on the same machine, the session cookies can be readily shared among them. In other words, if one logs on to the server via the account major or minor, the shared session cookies can allow him to access automatically the web scripts in the other account. The diagram for our design is depicted in Fig. 2 below.

Fig. 2. Dual-account design for better server security

We note that a page request can be sent to the server scripts of user major or minor, and the user minor can optionally host a parallel login support depicted in the dashed lines and box. Scripts-a and data-a are private to account major, while scripts-b and data-b are private to account minor. The “pub data” are generally readable to everyone on the same server machine. This diagram shows that the content on the account major will not have any security risks from executing the unsafe client programs on the account minor. We also note that one of our code evaluation system, evaluating students’ C++ programs, is first developed exclusively on just one account, but the production is made via the dual-account design as in Fig. 2. Hence this dual-account scheme is our next level of security protection. It has a much improved security at the mere cost of an additional regular user account. No resources or support of system administrators are required, and this is the main advantage and is our current preferred mechanism.

The third level of security protection is to utilise a dedicated computer machine for the server with complete system administration privileges. Fig. 3 illustrates such a design in which actions to be conducted by the root is shown in pink boxes with thicker borders. The execution and data on the minor account, the account of a typical registered user, are shown in light tortoise, and the rest are done by a designated user major or kept in its account. We note that the source analysis component S and the execution evaluation component T are exactly the same as in Fig. 1.

If a client’s program accidentally did any damages to the environment of his account, then the account may be automatically regenerated by an online request. In fact the system may even be designed in such a way that regeneration of all the client accounts is done periodically. We note that our proposed dual-account tier is very much like all the client accounts being merged into a single one, and a more regular regeneration of the account would almost achieve a similar effect as the multi-account tier. To conclude this section, we note that our proposed three protection tiers shifted from the minimal use of dedicated resources and privileges, at the cost of more stringent security check at the source code level, to the minimal security check on the user programs at the cost of dedicated computer server, full control privileges as well as dedicated network domain for the Internet.

IV. DELEGATE EXECUTION TO CLIENT SITES

There are many ways for a program source code to be evaluated through its execution, and some of them are already partially incorporated in the e-education [10]-[14]. Although all of program code can be executed on the server or another specifically designated machine or account, some may be delegated back to the client machines. In principle, the delegation is for the execution of the woven programs, along with the designated input data, and the output data of the execution will be sent back to the server for evaluation. Obviously it requires a proper design on how portions of user code are woven into a test program, what input data should be for the testing, and what is the expected output data. A client can be asked to compile the received woven source code before executing it, or could even choose to receive directly the compiled executable for the execution if the client considers the server trustworthy. In almost all cases, a client would need a separate software program to manage these tasks. A client can comfortably adopt such a program if it is transparent with its security details. However, a reasonable compromise is to make use of one of the standard software programs such as a mainstream browser, which most will feel trustworthy with its security implications.

To illustrate a simple execution delegation, it is easiest that we examine a case of evaluating an HTML web page with CSS and Javascript. The simplest case is perhaps to test if a web page has set proper hyperlinks, fonts and
layout via tags and CSS. For example, suppose a user is expected to create the following **user.html** as an assigned task

Sample Page for IE
<ul> UL ELEMENT</ul>
<li> hyperlink <a href="link.html" target="DO">anchor</a> </li>
<li> font bold </li>
<li> style color:red; border-style:dotted;CSS </li>
</ul>

Then the extraction of the assessment features inside can be achieved by appending it the following data extraction code

```html
<form name='pass' method='post' action='process.pl'>
<input type='hidden' name='p' value=''></form>
<script>
var s=n[1].childNodes[3].style;
p['color']=s.color;
p['border_style']=s.borderStyle;
var c=n[1].childNodes[1].childNodes;
p['color_name']=c[1].tagName;
var c=n[1].childNodes[2].childNodes;
p['color_name']=c[1].tagName;
s=c[1].style;
p['font_family']=s.getAttribute('fontFamily');
for(var k in p) q += k+'='+escape(p[k])+'; ';
document.pass.p.value=q;
document.pass.submit();
</script>
```

This way, the user code contains an expression that is equivalent to (or exactly the same as) `width*height` then the combined source code can be compiled and executed successfully with exit code 0 and without any output to the `stdout` device. If the user-entered expression doesn’t evaluate to the expected value, then the exit code will be 1. The compilation and the execution of the combined code can be done either on a client machine if client authenticity is not of a great concern, or on the server so that the evaluation results cannot be tampered by any clients. The output to `stdout` can also be collected, and the evaluation system may decide whether the unexpected output should be flagged as an error or it will be simply ignored. Since most programming languages are not designed for symbolic evaluations, we propose to adopt the principle of “impact equivalence” in that two code entities are considered equivalent if their outputs are identical for a sufficiently large number of random input data. Hence we may for instance utilize “enough” random samples to test the validity of the “symbolic” expression. For running unsafe Java codes, on the other hand, the dynamically administered security policy can be further utilized to secure the integrity of the server host.

We finally note that although it’s in general of similar level of implementation complexity for either the delegated execution on the client machine or the centralised execution on the server machine, it may be worthwhile to run the potentially faulty exercise code first on the client machine, especially for the purpose of client training. As to what extent the execution delegation should be sought, different levels of security concerns and convenience demands will lead to different strategies.
For a C++ programming drill system designed to support a university subject, for instance, we nevertheless run everything on the dual-account server so that evaluating both exercises and quizzes can be done online automatically. We are now also implementing an in-house online Java drills in a similar manner.

V. IMPLEMENTATION EXAMPLES

We now move to examine the implementation of our proposed security tiers on some applications. We will in particular examine the implementation of a programming drill system to automatically evaluate the student programming code while issuing error messages or other coding tips when pertinent. In this section, we will first address the design and security issues for the implementation on the first two security tiers, and then discuss these issues again when the implementation is to be made with the multi-account tier.

A. The Design Specifics

In a typical case, a user will be posed a question which describes what requirements are to be fulfilled, and then the user will submit a program segment to achieve the goal. Since in most cases such undertakings are much less prone to security concerns, we can easily implement it initially on a single user-account of a given Linux machine, using the cgiwrap scheme, and then alter it to the dual-account implementation for the final production.

We note that in Fig. 1, a typical benchmarking test $T(k)$ is composed of extracted parts $\{P_i\}$ of the client program and additional parts $\{Q_i\}$ from the test designer so that the combined new program $T(k)$ is to produce the expected output $O(k)$ on the prescribed input $I(k)$. For a simple client program, the $\{P_i\}$ are often just a single part, the whole client program itself. For our evaluation of program code, a benchmark test is typically written in the form of a script file, and the execution of this script will be regularly checked to see if it has terminated or not. If it is not terminated within a given period of time, it will be deemed that the user code contains an illegal infinite loop and the execution of the benchmark script will be automatically terminated forcefully, implying also that the benchmark has failed. Whenever a benchmark is aborted, further cleaning-up of the sub-processes created within the test will also be conducted.

For a code evaluation, we first conduct the source level partition or extraction of the client program $P$ into a number of constructs $C(k)$, if necessary, and then analyse them to see if anticipated program constructs such as a for-loop or a particular user function are present or suitable. If this part $S$ in Fig. 1 is completed successfully, the evaluation moves into the live testing $T$. For a different drill question, there may be different number of benchmark tests associated with it, and usually the first test is to see if the program can compile successfully. If one by one all the tests have been successfully passed, then the client program $P$ has fulfilled the requirements set out in the question, and the procedure will generate the benchmark report or score and complete the task. If a benchmark test times out, then the control returns to the action menu. If the test fails, then the corresponding analyser will be executed to generate a suitable error message or provide other form of feedback. It also allows the user to activate a compiler IDE with the complete program source loaded automatically so that the user can debug it within the IDE, which is essentially a security risk delegation to the client machine.

There can be various other auxiliary measures to enhance the security, when dealing with a very specific programming language. For the testing of program code, for instance, the client code $P_i$ is typically enclosed into a separate function. More implementation-specific details for the C++ drills can be found in [12].

The auto evaluation of program code is thus implemented for both C++ and Java in the form of a programming drill system, and was also used for the teaching purpose of an undergraduate unit called
programming fundamentals, see Fig. 4 for a typical user interface. It is perhaps interesting to note we can also provide students with an online C++ compiler at the cost of reduced keyboard interactivity, see Fig. 5, and this will enable students to practice their programming skills literally anywhere, possibly even on a mobile device.

**B. Compare with the Multi-Account Tier**

First of all, the programming drill system on the dual-account tier described in the previous subsection can also be implemented on the multi-account tier. For the multi-account tier as described in Fig. 3, one would need an account super of root status. All the scripts on the web server will still be executed under cgiwrap so that the permission status of that user is retained. This way the scripts under the user account major will be the entry point and will coordinate the data communications between the pages or scripts belonging to the different user accounts. The account minor, representing a typical registered user account among the many, will host the scripts to execute program code to be evaluated. This account will thus restrict the potential security damages done by the unknown program code to this regular user account only. However such security damages can be easily rectified by simply deleting the whole account and recreating it again with the standard set of data and support programs. The account creation and data movement need to be undertaken by the user super of root group, and it can be achieved through the URLs such as /cgiwrap/super/setup.pl where in the place of the script setup.pl one can also have the scripts to copy data or support program code from account major or account super, and have them owned by account minor. Moreover, the scripts in the account super will first validate that the request comes from the computer of the same IP address, and/or contains secret keys of other forms. The rest of the implementation is just similar to that for the dual-account tier.

In comparison with the multi-account tier, the dual-account tier can be regarded as grouping the security breaches of all clients together into the single account minor. If there is just one breach, then the account minor would need to be refreshed, i.e. recreated, while in the case of multi-account tier, only the user account responsible for the security breaches will need to be regenerated. We note it is also convenient to get the root to create a batch script that regenerates a given account and have certain scripts and data owned by the root user but readable and/or executable to the other account owners. This is to ensure that even rogue code won’t be able to alter the important scripts or data. In these perspectives we see that the choice between the dual-account tier and multi-account tier is largely a trade-off between convenience and effectiveness. If deliberately engineered rogue code is considered rare, then the dual-account tier is much simpler to host and maintain. The system “health” can be well kept by scheduled regeneration of the account minor, or simply by a manual regeneration on a need to basis. If the program code to be evaluated is expected to be routinely ridden with bugs and traps, then the multi-account tier would be more suitable, even though a dual-account tier with relatively frequent account regeneration will also fulfill the purpose. The main drawback to the use of multi-account tier is that it requires a dedicated computer with a system administrator assigned to it, and it will need to have the site specifically opened to pass through the corporate firewall. In short, it can easily become an undertaking beyond what for instance a coaching instructor or an errand techie is willing to do.

While our proposed three security tier solutions are not algorithmic in nature and a great many of factors related to hardware and software must be known before a somewhat quantitative comparisons can be made of their performances, we could nonetheless still roughly estimate their complexity in regard to the total number n of web users. In general, suppose the average computational load for each user request, say, executing a piece of submitted user code, has the quantum Q, then the complexity of processing n requests under both single-account and dual-account tier would be Qn, or just O(n), as the dual account authentication overhead is much less than Q and is negligible. For the multiple-account tier, since each user request generally needs to be accompanied by a user account annihilation, account recreation, and minimum system setup and data deletion and copying, the computational overhead R typically observes R\gg Q, leading to its upper complexity bound to (Q+R)n \gg Qn. Event though the resulting complexity is still at the same scale O(n) mathematically, under the assumption that the relevant numbers are still within the range of the hosting computer’s linear scalability, the difference in the actual amount is still large and the increase in magnitude more of computing load could run the risks of taking a given hosting system to beyond the range of linear scalability. All in all from the pros and cons we discussed in this subsection, the resource limitation on the hardware and technical know-how will find the dual-account tier implementation more desirable as is the example in this section, particularly for the popular type of technical infrastructure a small to media-sized company or a learning and training institute has.

**VI. CONCLUSIONS**

We have proposed three protection tiers of different security strength and resource drain to cater for the needs to execute and evaluate real-time the untrusted computer programs on a Linux-based web server. While the multi-account tier can both sustain the potential security attacks by the untrusted code and better isolate the origin and the affected area of the attacks, this tier may often require resources beyond a casual developer. The single-account tier is the least resource hungry, but it requires heavy screening at the level of source code and may result in some cutbacks on certain programming power. The dual-account tier is a balanced compromise in that it doesn’t require much additional resources, and yet it can tolerate...
the attacks as much as the multi-account tier, and the somewhat downgraded preciseness to locate the attack source and affected area is not really of any significance particularly when rogue program code is expected to be rare. Finally a programming drill system aimed at training the university students is implemented in the context of these different protection tiers, with their pros and cons compared in regard to the implementation.

REFERENCES


Zhihan Jiang received the B.Sc. from Zhejiang University in 1982, and Ph.D. from the Victoria University of Manchester, Institute of Science and Technology, UK, in 1987. He is currently affiliated with University of Western Sydney, in the School of Computing, Engineering and Mathematics. His pertinent research interests include mathematical modelling and algorithms, web based security and applications, as well as image and video processing.

Jiansheng Huang received the B.E. and M.E. from Hefei University of Technology in 1982 and 1984 respectively, the MSc from The University of New South Wales in 1997 and the PhD from The National University of Singapore in 1999. Currently he is working with the School of Computing, Engineering and Mathematics. His relevant research interests include information systems and security, power system operation and protection.

Rezina Akhter received Master of ICT from the University of Western Sydney in 2013. She is currently working as a postgraduate research student at the School of Computing, Engineering and Mathematics. Her current research lies in the field of knowledge-based auto-training systems, and web based secure information systems.