Moving Target Defense Against Injection Attacks

Summary:

There are various techniques discussed in the paper by the author to prevent SQLIA's such as taint-tracking, static and dynamic analysis. But in this paper, they have tried to disrupt the SQL injection attacks by making advancement on a static mutation technique by mutating various attributes (such as programming language and different type of database) on a periodic time basis. The technique is based on Moving Target Defence called dynamic defence to SQL Injection attach (DTSA). A mutation strategy is first used to determine the attributes which need to be diversified. Once the attributes are determined the frequency of mutation is turned into focus. The optimal frequency of mutation is decided based on the number of times the attacker was able to get exposure to attributes, by applying the concept of dynamic programming.

Pros:

- Although the workload is increased by applying DSTA it provides an acceptable workload performance compared to other mutation technique.
- Dynamic programming is used to find the optimal time of mutation. So when the attack is sensed the mutation time period is shortened which can effectively slow down the process of the attack.

Cons:

- DSTA is applicable to relational databases only and not NoSQL databases.
- A separate server machine called core unit is used by DSTA which decided which mutant should be selected based on a certain amount of periodic time.
- The DSTA architecture will randomly select the programming language and the database so the user is served at different rates based on the programming language and database performance.
- Same web service has to be implemented in various language and the same data is to be stored in various databases and maintained to be served as mutants.

Bringing the Web up to Speed with WebAssembly

Summary:

Based on the necessity of the hour the authors from the 4 major browser vendors united to provide a common solution for high-performance web applications. The authors in this papers discuss the language called WebAssembly which is the recent solution for cross-browser fast low-level code. WAMS is a new portable code format to bring a native code performance in the browsers that are being used on a daily basis. WASM is developed to provide high performance and to librate web from javascript. Before WASM there were many alternatives such as asm.js and Portable Native Client which were working in the space to make the web exposed to native speed but now these projects are merged to work on one goal that is WebAssembly.

The authors in the paper talk about the basic point such as the format of Webassemly which is a byte code for making the language compact, portable and designed for jitting not interpreting. Computation model is more like that of a stack machine which is static in nature to represent the code in a compact manner. The authors also talk about some surprising design choices such as linear memory. In WASM there is no concept of built-in objects but it consists of just a byte array which uses the integers as pointers the instruction similar to hardware are present to perform all the operations on the memory. There are no arbitrary jumps in WebAssembly the code in it made of blocks and break. The breaks allows you to break out of the block or nested blocks. But when the irreducible code is expected in the WebAssembly Relooper algorithms is expected to be used by the producers. For safety, WASM is type-checked. WASM programs are organized or structure into modules. Modules have the functionality of import and export. Using imports WASM can host functions for Web API written in Javascript.

Pros:

- WASM is not proprietary language.
- WASM is designed to enable the feature on the Web which Javascript also does not support.
- WASM does not have an irreducible control flow. It has blocks and breaks which allows the language to be fast to validate and compile.
- The first language that provides the proof of soundness which is machine-verified.

Cons:

- Only support for web technologies and currently does not have any support to desktop or standalone applications.

- Currently, no support for high-level languages as there is no support for concepts like garbage collection , exceptions, threads, SIMD, closures, stack switching and multiple return values.

Wasabi: A Framework for Dynamically Analyzing WebAssembly

Summary:

WebAssembly has many use cases such as audio/video processing, machine learning at the client-side or full 3d games. So to progress with this new platform there is a need for dynamic analysis tools to ensure the correctness, performance and security of future web applications. Dynamic analysis frameworks have been tremendously had been the basis for providing the framework for such tools example pin,valgrind, roadrunner and jalangi.

Wasabi aims to be a generic platform for heavy-weight dynamic analyses for web assembly. It allows observing all instructions with inputs and outputs all web assembly instructions with inputs and outputs. Example usage of wasabi usage is given in the paper by the author which is instructions profile for crypto miner detection.

Wasabi operates in two phases
Instrumentation phase: In this phase, it instruments the binary to get the instrumented program and also generates some auxiliary information.
Analysis phase: During analysis at runtime, the instrumented program calls the client analysis via some low-level hooks additionally providing some meta-information about the execution.

Pros:

- Wasabi was the first dynamic analysis framework for WebAssmebly before this there was no general-purpose dynamic analysis framework.
- Wasabi provides easy-to-use and high-level APIs to implement heavyweight analysis that can monitor all low-level behaviour of the WebAssembly program provided.

- The framework covers the problem of tracing polymorphic instructions with analysis functions that have a fixed type via an on-demand monomorphization of analysis hooks, and also statically resolve the relative branch labels in control-flow constructs during the instrumentation.

Cons:

- Wasabi only provides a dynamic analysis of WebAssembly currently, still the framework needs to support the cross-language dynamic analysis of web applications running on Javascript and WebAssembly is a future thought according to this paper.

On the Effectiveness of Address-Space Randomization

Summary:

The paper discusses the strengths and weaknesses of the Address Randomization techniques in systems having operating systems like Linux and OpenBSD having a 32-bit architecture which has some limitations due to less number of bits for address randomization.
The author in paper discusses weaknesses of ASLR and shows with an example of a return-to-libc attack that how the ASLR can be disrupted. PaX ASLR used in the Linux and OpenBSD operations is used while performing the experiments in this paper.

The attack used in the experiment of this paper is not the usual return-to-libc attacked in fact it is a chained return-to-libc attack. Traditional Return-to-libc attack which has both the information related to stack and text segment of libc is not used to exploit the address space in the system. But in the technique used in this paper uses only address information placed by the target program on the stack. A simple experiment to show that a derandomized attack takes a long time such as 216 seconds to exploit the Linux PaX ASLR system. The author also explains the related work done on write or execute pages a security feature to counterattack the various attack like return-to-libc, as under the “write or execute” the pages in various types of memory segment is writable or executable pages but not both. But due to this patch, there are high-performance penalties.

The author then also discusses the improvement that can be made in the ASLR to fortify the chances of preventing the program from various attacks. The authors suggest improvements such as by upgrading the system architecture to 64-bit s it gives more address space(for example 40 bits mentioned in this paper) for randomization on 64-bit architecture rather than a 32-bit architecture which gets only 16 bits. To increasing randomization frequency of the

address space(randomisation need to be done for every attack on the address space) with each attack as it will further slow the process of attack being launched on the address space. The author also suggests finer granularity randomization, both at compile-time and runtime. At compile-time, the finer granularity can be achieved by randomizing the variables and functions by doing simple modifications to compiler and linker or by introducing random padding into stack frames. At runtime, the finer granularity can be achieved by randomizing more than 16 bits of address space, reordering functions. The fourth way which the author discusses to improve the address space randomization is to use a watcher which is a system service that monitors the actions of attacks and shut them down after some threshold. At last, the author suggested using overflow mitigation systems such as StackGuard, Propolice and PointGuard which are independent of the Address space layout randomization and which can disrupt many exploits including the buffer overflow is another technique discussed by the author which can be used with ASLR to defeat many attacks.

Pros:

- Effectiveness of the ASLR can be increased by applying various techniques suggested by the author in the paper such as increasing the frequency and granulation of randomization.
- The author in the paper suggests that the randomizations techniques can be combined with watch daemons (services which monitor for any attacks) can be used to prevent return-to-lib attack mentioned in this paper and denial of service attacks.

Cons:

- ASLR is machine-dependent as the author in the paper suggests using 64-bit machines over 32-bit machines, as ASLR is more prone to brute force attacks in the latter types of machines.
- Compile-time randomizations may lead to exposure of sensitive data in the code.

Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks

Summary:

In this paper, the author explains how a binary code injection attack can be disrupted by a prototype based on Valgrind x86-to x86 binary translator. The author discusses the implementation and limitation of RISE and discusses how the RISE disrupts the various types of attacks. The RISE is an obfuscation technique used to protect the program by scrambling the binary code by updating instruction with a constant unique seed for each program execution.

Without the knowledge of the key, the attack code will be processed and when it gets descramble the code will crash.

When using RISE as a binary-to binary translator, to avoid the execution of malicious code injected in the form of data in the currently executing code adoption of a method such as code shepherding is used. There are two security policies which come under the method of code shepherding such as code origin pollicies and restricted control transfers. These policies enforce security by optimizing interpreters.

One advantage of RISE over avoiding the buffer overflows that is not necessary to apply it on the whole system like PaX. But it can be applied to the selective processes running in the system. The author of the paper considers 14 types of attacks and verifies the impact on Valgrind with RISE and Valgrind without RISE. Valgrind with RISE avoids number of attacks including the Synthetic heap, stack overflow, Bind NXT and SAMBA trans2 attacks.

In terms of performance, Valgrind with RISE performs 5% slower than the Valgrind without RISE, as there is more instruction in the former than the latter.

Pros:

ors raise the point on how diversity is overlooked in the mass market of computer systems

Cons:

ors raise the point on how diversity is overlooked in the mass market of computer systems

---

Building Diverse Computer Systems

Summary:

The authors raise the point on how diversity is overlooked in the mass market of computer systems and compares it to the dominant species of society. The authors try to convince as in our society how diversity is important such importance of diversity is also needed in computer systems. The author discusses various methods for achieving diversity based on randomizations with a focus on computer security.

Few of the randomizations methods mentioned in the paper by the author:

1. Adding or deleting nonfunctional code
   Insert or delete a no-ops instructions at random locations in the compiled code which. This technique has the ability to disrupt the Kocher's timing attack on RSA.
2.

A. Reordering code-

Basic blocks can be reordered in a random way and stored at different locations but the order of execution is not affected. But this technique does not provide security against a large class of viruses.

B. Optimizations for parallel processing:
Code is divided into blocks of instructions and executed on multiple processors simultaneously. We need to select the code which is intended to be run on a single processor. But the amount of randomness that can be produced with the method is limited to the no of processors being used.

C. Instruction scheduling:
Padding on stack frame by a random amount
Randomize the locations of global variables and also offset assigned to local variables.

3. Memory layout:
1) Padding each stack frame with random gaps in memory.
2) Randomizing the global and local variables
3) Assign the memory to the stack frame in randomized manned not in a contiguous manner.

Some Other Transformations that can be done for Randomizations
Process Initialization
Dynamic libraries and System calls
Unique name of system files
The magic number in certain files
Randomized Runtime checks

Pros:

- Diversity techniques discussed in this paper help in developing software systems which are more robust and immune to various exploitable holes or attacks.
- Also, the diversification might help in curbing the replication of attack on different yet exactly the same system software.
- Bugs in the software system can be detected by applying the above-mentioned diversity techniques in the early stage of the software development process.
- Attacks like convert channels

Cons:

- Applying diversity technique is not only the solution to solve all the vulnerabilities, but also cryptography techniques should be applied hand in hand to counter various security issues.

Preventing overflow attacks by memory randomization

Summary:

The author in the paper talks about two approaches for preventing overflow attacks by Memory Randomization. Both the approaches are implemented at the user's end.

1. Fine-grained stack randomization to avoid stack smashing overflow attack: Done by randomizing few of binary instructions of the stack.
2. Fine-grained heap randomization

The Stack randomization is not done at the time of compilation rather it done at the time of installation of the program at the user end and it done at various levels. Below are the randomization techniques used
Addition of dummy variables during a sequence of program instruction compilation
Padding the stack frame.
Randomizing code at the level of assembly instruction level with help of assembler

But the difference between the approach in the recent research randomization and the one which they have implemented in this papers is the approach taken towards the randomizations of the stack frame. Fine-grained stack randomization techniques used in this paper, which was performed on various binaries where the routines are randomized independently of other routines in the binary
Advantages of this technique that it is low cost and provides an easy way out for the applying randomization on binaries instead of large scale applications such as linker/compiler/loader.

A technique for fine-grained heap randomization is also presented by the author in this paper. The library patching technique used in this solution adds random pads above and below every chunk of memory allocated off the heap. The values for these random pads are generated at run-time due to which each instantiation of a program allocates.

Pros:

- Randomization definitely increases the effort of the attacker in creating an attack.
- Stack frame randomization technique discussed in this paper decreases the probability of being hit with a stack smashing attack.

- The fine-grained stack frame randomizations actually take less load than the already existing techniques.
- In heap randomization, the randomizing is done at the runtime and it does not introduce any runtime overhead time when the application is being executed.

Cons:

- Although Fine-grained stack randomization tries to disrupt the buffer over flow attacks it does not guarantee protection against the attacks which corrupt the data adjacent to vulnerable binaries.
- In windows operating system a separate patch library is to be prepared and injected to provide the functionality of heap randomizations as it is a proprietary operating system.

Online videos on WebAssembly(Just for reference)

Video - WebAssembly: Binary in Plain English by Milica Mihajlija at DACHfest 2018

Questions Answered:
What it is?
Js is not is ideal for all task so we need something new that is WebAssembly to increase the performance.
WASM is used to increase the performance on the web.
WASM is in Binary format so fast to load and fast to execute
Another high-level language is compiled to web assembly. Such as C++,C and Rust
Web assembly is not assembly language as it is not for any specific system but for the browsers.
Not a replacement to js but used with js to increase the performance of the online applications.

Why do we need it?
For Performance
Online Game need a high degree of performance
The more rich application came to the web which cannot be handled by the js so we need something more that is WebAssembly.

How you can we benefit from it?

WebAssembly brings speed(small binaries), portability() and flexibility

Speed
Js - parse, compile and optimize the code as it is being executed on-page.
WebAssembly- delivered as binary so the decoding happens much faster.
WASM Statically typed so no overhead of which types are going to be used
No garbage collection in WASM
20% slower than native code
Portability-
Whenever a code is running on the browser it has to be compatible with device processor architecture and the operating system. WASM needs only compilation step and the source code will run in every browser

Flexibility-
Lot more developer can code on the Web.

How does it work:
LLVM compiler outputs Web Assembly
Emerscripten- based on LLVM for compiling C and C++ to WASM
Rust has its own compiler called rustc to convert the source code from Rust to WASM.

WebAsembly does not access with DOM, WebGL, Web audio other APIs so to work with this we need the emerscrpiten and rustc will give output in form of WASM+js+html. So that these apis can be accessed from the js file.

Web Assembly functions can be called from JS code.

Buffer Overflow Attack:
A buffer is a place where the data is stored informs of the blocks.
When the buffer is overflown with data, the overflown data can move into another buffer.
Moreover this
There are two types of buffer overflow attacks:

1) Stack overflow attack
2) Heap overflow attack

WebAssembly Demystified:
Efficient Low-level byte code for web
Fast to load and execute
Byte code- 0x6a- 1byte it is an operand of wasm
Safe and portable same way was js.
Wasm is the shortcut to your js engines optimizer
Is it going to kill Javascript? No, maybe someday when it supports all language

JS is an extremely dynamic language
Turboscript

WASM is stack machine language
Stack machine: instructions on the stack
132.add - > 0x6a

I32.const 1
I32.const 2
I32.add
call $log --returns the result of add

One more representation of WASM is using the Abstract Syntax tree AST(but compiled and evaluated as a stack machine)

```
(call $log
        (i32.add
                (i32.const 1)                    it looks like lisp
                (i32.const 2)
        )
)
```

Source map support is likely coming eventually

How I get started?
WebAssemby explorar

Webpack is planning to add support

What's missing?
Direct access to Web Apis, right now js should be called.

Garbage collection:
No garbage collections not present in the wasm.

Multithreading is next

Web assembly is going to be close to native threads

WebAssembly: Expectation vs. Reality (2-hour practical workshop)

What is WASM?
Byte code for the browser
All major browser vendors agreed on the format

What languages can be compiled?
https://github.com/appcypher/awesome-wasm-langs

Memory:

GPU:
No GPU access
Not a priority for WASM

Host Binding

Limitations:
No concepts of Threads, Exception Handling

Limited no of Reference type