3/1 Web Security

Scribe: Saharsh Agrawal
Papers: Privilege separation in HTML5 applications and Securing the Tangled Web

Discuss Summary Question about Auto-Escaping

- Why not have template systems always do contextual auto escaping for everything, but developers can opt out of auto escaping from certain parts?
- Why is there a need to opt-out at all?
    - There may be web applications that want to use a different sanitization policy than the auto-escaping would.
    - Example: Blog with ads --> ad provider says come HTML/ Javascript code needs to be pasted into every page of blog
- Template systems avoid the challenges of static analysis; it can be hard for static analysis to find all the data flows through the application , but in contextual auto-escaping, escaping is done at the output (in the template.
- The contextual information needed is not about where the data came from or the previous data flow, but about where the data lives inside the html document.
    - Auto-escaping doesn't need to do static analysis
- Rendering markdown to html
- Why not just let the developer opt-out?
    - Developers in a hurry might just turn off auto-escaping, and this could go undetected.
    - Developer may try to do their own auto-escaping/ sanitization but if they do it wrong, now the code could be vulnerable.
        - Example: Django has auto escaping with opt-outs. Out of 7 web-apps examined, 2 of them had XSS vulnerabilities because the developers had turned off auto-escaping to do their own.
- Template systems
    - Unsafe by default: didn't do any auto-escaping for you
    - Safe by default: auto-escape everything, but with opt-out possible
    - Safe by construction: trying to head towards a system where no matter what developers do, it will be safe
- Connection to taint tracking
    - Effectively, they are tracking the taint status of the data with the type of that data:
        - Anything of type string is considered tainted and as coming from an untrusted source by the templating system.
        - Anything of type safe HTML is considered untainted and safe by the templating system
    - The type system ensures the properties you want from taint tracking; anything that came from the user should be considered unsafe and tainted.
    - User can send strings, but cannot send things of type 'safe html'
    - You could do taint tracking by augmenting the type system with separate types for the tainted values and the untainted values.

- During bundling, maybe chrome could modify the bundle to automatically add shims, and stuff
    - Partitioning code between parent and child may be tricky. Can we automate this partitioning? Runtime monitoring or static analysis to take an extension and come up with a parent-child partitioning. One heuristic might be: any code that is accessing privileged resources, will have to go in the parent.
    - However, it could get tricky. For example, in the ScreenCap extension: only take a screen capture if the user clicks a visible button in the UI —> parent not only needs code for screen capture, but also the UI code for the button. If we put the UI code in the child, we would end up with an insecure privileged separated version of the code.
    - Privilege separation done in other areas (not necessarily in the web context) e.g. Intel SGX.
- Privilege separation does not do anything about malicious extensions, but just limits the harm that can be done. A malicious extension could put all of it s code in the parent, and the parent has full privileges.
    - Developer takes measures so that if there is a bug in the code, the impact will be reduced.
    - “Benign but fallible developer”

<u>Emma Lurie Discussion</u>

- Magic quotes - PHP
    - When on, all single-quote, double-quote, backslash, and NULL characters are escaped with a backslash automatically.
    - Goal is to prevent SQL injection attacks
- Would magic quotes succeed in preventing SQL injection attacks?
    - Fact 1: Magic quotes could be turned on or off in php.ini
    - Fact 2: Magic quotes use the generic functionality provided by PHP’s addSlashes() function which does not correctly handle Unicode
        - Code might get escaped that shouldn’t get escaped. For example if you have some unicode symbol which has multiple bytes and one of those bytes happens to match the byte for a quote or a backslash, then it will get escaped even though it shouldn’t.
    - Fact 3: Recent versions ofMySQL require single-quotes to be escaped with a second single quote, not a backslash
- Even with magic quotes, there is still room for SQL injection attacks.
- Magic quotes were designed with the non-expert developer in mind; kind of like a quick and dirty security check.
- Took a number of years to get rid of this feature (now deprecated, but broke existing PHP code because it was widely used)

<u>Parse-Trees Instead of Text</u>

- Today’s world: Servers build up HTML documents as strings, and then that HTML document is sent as text content down to the browser. Consider if the webpage was instead transmitted as a parse-tree.
- And on the server-side code, when we write the server-side code, we write it not as working with texts, strings, but we write it as operating on these parse trees.
- Is it still vulnerable to XSS in the parse-tree world?

    - The tree still has to get sent to the browser as a sequence of bytes after being serialized. On the browser, it is deserialized and converted back into a tree. As long as there was safe serialization and deserialization code, this is probably safe. But, if the serializer was not careful about how the tree was encoded into bytes, this could introduce the same problem.
    - Relation to contextual auto-escaping:
        - Think of contextual auto escaping as an attempt to emulate the world where everything is done with this tree-based approach
    - If we only ever worked with trees and never with strings, and the serialization and deserialization was correct,, then it could be “Safe by Construction” and we would not have XSS bugs.
- However, we are not in the parse-tree word; instead, we work with text and strings, since the interface between servers and browsers is text —> this is what results in XSS bugs
    - Switch everyone over to using trees? Not realistic
    - Server will unparse the tree at the last step and turn it into HTML text.
    - Programmers like working with text; writing everything in tree format would be hard
        - Solve this by using templates on the server
        - Template system is like a compiler; lets us write the template in a convenient markup format which will then be converted into code that uses the tree-based programmatic operations on trees which are safe.
- Lesson: work with pre-passed objects, not strings; this eliminates a whole class of vulnerabilities related to injection and parsing errors.

SQL Prepared Statements

- Bad: ’SELECT * FROM Users WHERE username=‘ + user
- Good:PreparedStmt(’SELECT * FROM Users WHERE USERNAME=?’, user)

Blueprint

- Proposal whose goal is to not have to place as much trust in the unparsing on the server side
- Emulate the template system without having to know how to parse.
- Server will send HTML document (the template) which has dynamic values replaced with placeholders
- Then the server will send some static javascript code which will get the relevant tag, and call setAttribute to replace the placeholder value.
- The server is going to send to the browser a document that looks something like this:

```
Blueprint:
<img src="x" id="1"></img>
<script>
getElementById("1").setAttribute('src', '...');
</script>
```