**CS378 Ethical Hacking**

**Final Project**

##### Alex Irion, Tony Sampson, Krzysztof Drewniak

##### What is the tool?

##### Our tool, elit3\_h4ck.py, performes an automated phishing/XSS attack. As a hypothetical, we supposed that the UT Login page (<https://login.utexas.edu/>[)](https://login.utexas.edu/)) had an XSS vulnerability. Since it does not, we cloned the page and hosted a copy of it that contains an added reflected XSS vulnerability on the cs378-ethical-hacking. Ideally, we would host our cloned site where it is publically viewable, but for this proof of concept the site can only be accessed from the cs378 network.

##### The introduced vulnerability consists of a base64-encoded parameter to the login page being decoded and echoed back without proper escaping as the value of a hidden form field. This is a simplification (for practicality reasons) of a vulnerability that could hypothetically exist in the UT login system as designed. The UT login system’s forms take one GET parameter, SAMLRequest, which contains XML conforming to the SAML specification which is then compressed using deflate and then base64 encoded. For example, the decoded value of the SAMLRequest parameter from an attempt to log in to Canvas is:

##### <samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" ID="ceee158682cbb06d6031c1acce2f0c7c335874f783" Version="2.0" IssueInstant="2017-05-07T19:53:44Z" ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST" AssertionConsumerServiceURL="https://utexas.instructure.com/login/saml"><saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">http://utexas.instructure.com/saml2</saml:Issuer>

##### <samlp:NameIDPolicy xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" AllowCreate="true"></samlp:NameIDPolicy>

##### <samlp:RequestedAuthnContext xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" Comparison="exact"><saml:AuthnContextClassRef xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml:AuthnContextClassRef></samlp:RequestedAuthnContext>

##### </samlp:AuthnRequest>

##### We supposed that, for example, the AssertionConsumerServiceURL could have been echoed back into the login form decoded for some reason, in order to justify our simplifications.

##### After the vulnerable “real” website is in place, it is possible to perform attacks using it. The attack is designed to be directed at current Computer Science majors at UT, but can easily be modified to target other groups. The phishing email we send resembles a “Secure Academic Notice” notification email commonly sent by an automated system at UT. In order to induce users to click the link that causes them to log in to an exploited login form, the email advertises a scholarship award. The “From:” address in the email header is spoofed to “[san@utlists.utexas.edu](mailto:san@utlists.utexas.edu)”, which is the email address from which such notices are sent.

##### The exploit that is embedded in our email operates as follows. After escaping from the <input> tag, it inserts a JavaScript fragment that defines a function that, when called, POSTs the contents of the login form to an “evil server” and then calls the original onclick handler for the submit button on the login page. Then, the onclick attribute of the submit button is changed (once the DOM is ready) to point to the malicious function. The “evil server” is a small script that receives POST requests containing the expected credentials and prints them to standard output. This simulates malicious use of the captured credentials, since we’re not evil.

##### The exploit code and emails by populating a template with the current IP address of the machine running the hacking tool (which is also che machine which runs the credential-capture server) and the base64-encoded exploit code, respectively.

##### The script contains multiple methods of automatically obtaining addresses to send phishing emails to. It is possible to provide a list of emails to phish directly. You may also provide a list of names of UT students, which will be queried in the UT directory, where their emails are listed. Finally, you may provide a course ID from Canvas. If a course ID is provided, the emails of all students in the course are obtained from the Canvas API. Once emails are gathered, the phishing email is constructed and sent through a permissive SMTP server that allows forging “From:” headers.

##### Due to the nature of this exploit, and the possibility that someone might fall for this, our “real” website has been modified so that, after someone clicks “Log in” on the “real” login form, they are taken to a page where they are notified they have been phished. This page informs them that their password has not been stored and will not be used maliciously, but that they might want to change it anyway.

##### /Users/alexirion/Desktop/Screen Shot 2017-05-04 at 12.58.22 PM.png

##### [TODO, screenshot of phishing notification here]

##### /Users/alexirion/Desktop/Screen Shot 2017-05-04 at 1.01.52 PM.png

##### How do I use the tool?

##### Our tool can be run 3 different ways:

##### python elit3\_h4ck.py –efile emails.txt

##### The file ‘emails.txt’ provided should be a list of email addresses, one per line. Running the script with the –efile option doesn’t do any email collection since the file has all of the direct addresses we want to send our attack to preloaded.

##### python elit3\_h4ck.py –nfile names.txt

##### The file ‘names.txt’ provided should be a list of names of UT students, faculty, and/or staff separated by a newline. The script, when run with the –nfile option, will query the UT directory service (https://directory.utexas.edu) for each name in the file. The script will then scrape the returned pages in order to obtain an email for each listed name. This list of emails will then have phishing emails sent to them.

##### The names in the file must be found in the UT directory service.

##### python elit3\_h4ck.py –cid [id]

##### The ID provided should be a valid Canvas UT course id that the person running the attack has access to (has membership of). The person running the script will first need to obtain an OAuth token from Canvas for 3rd party application access (one is included).

##### The script will retrieve the emails of all students, professors, and Tas in the given course using the Canvas API

##### This form of the attack is directed at people in one specific class.

##### Tech specs

##### Cloned login.utexas.edu from the firefox web browser

##### Altered website to include PHP code that contains an XSS vulnerability

##### Hosted website on Virtualbox apache web server inside of cs378-ethical-hacking network

##### Python script to run exploit (python version 2.7.10)

##### Collects email addresses from UT directory web scraping or the Canvas API

##### Sends email through hushmail SMTP server that contains an XSS exploit

##### Starts server listening for POST requests containing credentials

##### Problems we encountered

##### Rate limited requests to UT directory, I even got completely blocked after too many requests

##### Canvas will only give email addresses of people in courses that you are currently or formerly enrolled in

##### login.utexas.edu didn’t have any XSS exploits that we could find, so we had to introduce one artificially

##### Hushmail suspended us after a while, so we had to make a different account

##### What could have been improved?

##### The body of the phishing email could have contained a more targeted attack, since we know that we are sending the email to other students in our class. The email could have contained information about a relevant Google doc, an assignment submission, a canvas message notifications, etc.

##### We could have publicly hosted our fake website so students don’t have to be on the cs387-ethical hacking network to access it (we were lazy and financially limited)

##### We could have made our cloned website look more legit by getting a certificate, real URL, etc.

##### We could have made the XSS exploit more realistic